

IMT School for Advanced Studies, Lucca
Lucca, Italy

**Fine-grained symbolic verification
of heterogeneous systems**

**Systematic exploration and formal analysis
of “what-if” attack scenarios**

National Ph.D. Program in Cybersicurezza
Track in Software, System, and Infrastructure Security
XXXVIII Cycle

By

Matteo Paier

2026

The dissertation of Matteo Paier is approved.

PhD Program Coordinator: Prof. Mirco Tribastone, SySMA Research Unit,
IMT School for Advanced Studies Lucca

Advisor: Prof. Marino Miculan, Department of Mathematics, Computer
Science and Physics, University of Udine

Co-Advisor: Prof. Gabriele Costa, SySMA Research Unit, IMT School for
Advanced Studies Lucca

The dissertation of Matteo Paier has been reviewed by:

Prof. Riccardo Focardi, Department of Environmental Sciences, Informat-
ics and Statistics, Ca' Foscari University of Venice

Prof. Michele Loreti, Computer Science division, School of Science and
Technology, University of Camerino

IMT School for Advanced Studies Lucca

2026

Contents

List of Figures	ix
List of Tables	xi
Acknowledgements	xiv
Vita and Publications	xv
Abstract	xix
1 Introduction	1
1.1 Problem statement	5
1.2 Research questions	7
1.3 Contributions	9
1.4 Structure of this thesis	12
2 State of the art	15
2.1 Verification of protocols	15
2.2 Interface-based modelling	19
2.3 Publish-subscribe modelling	21
2.4 Service meshes	26
2.5 Stateful policies	28
3 Introduction to ProVerif	31
3.1 Terms and processes	32
3.2 Security property analysis	34
3.3 Example: Diffie-Hellman key exchange	37

4	Interface-based threat models of coupled systems	45
4.1	Interface-based modelling	46
4.2	Running example: CIE	48
4.2.1	Protocols	56
4.3	Threat model	64
4.4	Restricting the search space	67
4.5	Phishing attacks	70
5	Fine-grained formal analysis of protocols	71
5.1	Modelling CIE protocols in attack scenarios	72
5.2	Modelling security properties	74
5.3	Dynamic generation and exploration of attack scenarios	75
5.4	Analysis results	77
5.4.1	Level 2 protocols: L2SMS, L2PSH, L2QRC	77
5.4.2	Level 3 protocols: L3APP, L3DSK	82
5.5	On the scope of symbolic modelling and analysis	85
5.6	Protocols improvement	88
5.6.1	L2SMS*: L2SMS improvement	89
5.6.2	L3APP*: L3APP improvement	91
5.6.3	L2PSH*: L2PSH improvement	91
5.7	Resource utilization of the CIE analysis	95
6	Extending interface-based models to decoupled systems	99
6.1	Pub/sub Dolev-Yao model	101
6.2	Refined architectural model	103
6.2.1	Differences with established models	105
6.3	Interface compromise	108
6.4	Running example: ABCAST with logger	112
6.5	ProVerif formalisation of ABCAST	115
6.6	Verification	117
6.7	Results	120
7	Improving system architectures with service meshes	125
7.1	Service meshes	125
7.2	Stateful access policies	128
7.3	Service mesh implementation in ProVerif	130
7.4	ABCAST with service meshes	132
7.4.1	Results	135
7.4.2	ABCASTSM*, ABCASTSM improvement	139

8	Implementation of stateful policies for service meshes	141
8.1	Security policies in distributed applications	142
8.2	OWSM: OPA Wrapper State Manager	146
8.3	Experimental evaluation	150
8.3.1	Research questions and methodology	152
8.3.2	Results	153
9	Conclusions	157
9.1	Contributions summary	157
9.2	Future work	162

List of Figures

1.1	A ROS system with two AVs.	4
1.2	Flowchart for the proposed methodology.	10
3.1	Syntax of ProVerif processes.	34
4.1	System interface model example.	47
4.2	Official documentation CIE interfaces.	50
4.3	CIE system interface model (initial).	50
4.4	CIE system interface model (first refinement).	52
4.5	CIE system interface model (second refinement).	52
4.6	CIE system interface model (final refinement).	53
4.7	CIE system interface model (further refinement).	54
4.8	Sequence diagram of the L2SMS protocol.	57
4.9	Sequence diagram of the L2PSH protocol.	59
4.10	Sequence diagram of the L2QRC protocol.	60
4.11	Sequence diagram of the L3APP protocol.	62
4.12	Sequence diagram of the L3DSK protocol.	63
4.13	Security implication between access control levels.	68
5.1	Pseudocode for the dynamic exploration of attack scenarios.	76
5.2	Overview of the L2PSH attack in the empty scenario.	81
5.3	Sequence diagram of the L2PSH* protocol.	93
5.4	L2PSH* protocol attack in the $\{A_{in:RW}^{mob-hum}\}$ scenario.	96
5.5	Computational resource footprints across CIE protocols.	98
6.1	Pub/sub Dolev-Yao model.	102
6.2	A generic hyperarc.	104
6.3	Summary of different interface models in literature.	106

6.4	Pseudocode for the ABCAST protocol.	113
6.5	Interface model for ABCAST with logger.	114
6.6	Sequence diagram of the ABCAST with logger protocol. .	115
6.7	ABCAST protocol attack in the $\{\mathcal{A}_{out:RW}^{r1,s}\}$ scenario.	121
6.8	ABCAST protocol attack in the $\{\mathcal{A}_{in:RW}^{t,r2}\}$ scenario.	123
7.1	A generic service mesh representation.	126
7.2	Interface model for ABCASTSM.	134
7.3	Sequence diagram of the ABCASTSM protocol.	135
7.4	ABCASTSM protocol attack in the $\{\mathcal{A}_{in:RO}^{t,r2}, \mathcal{A}_{out:RW}^{r3,l}\}$ scenario.	138
8.1	OPA's decision flow.	143
8.2	Diagram of OWSM implementation.	149
8.3	OPA vs OWSM: time performance.	154
8.4	OPA vs OWSM: concurrency scalability.	155

List of Tables

4.1	Number of scenarios for each CIE protocol.	66
5.1	Results for Level 2 CIE protocols.	78
5.2	Results for Level 3 CIE protocols.	83
5.3	Results comparison with the improved CIE protocols. . . .	90
5.4	Performance metrics for CIE Level 2 and 3 protocols. . . .	97
6.1	Implications at various levels of access compromise.	111
6.2	Automatic verification results for the ABCAST protocol. . .	120
7.1	Results comparison of the ABCAST protocols.	137

List of Listings

3.1	ProVerif code for the unauthenticated DH protocol.	39
3.2	ProVerif output for the unauthenticated DH protocol.	41
3.3	ProVerif code for the authenticated DH protocol.	42
3.4	ProVerif output for the authenticated DH protocol.	43
6.1	ProVerif code for the topic process for our architecture.	116
7.1	Different service meshes implementations.	130
7.2	Stateful policy service mesh implementation.	133
8.1	Example of Rego policy for RBAC.	145
8.2	Input for the RBAC example.	145
8.3	Data for the RBAC example.	146
8.4	Output for the RBAC example for the provided input.	146
8.5	Rego code for the token counter example.	150
8.6	Rego code for the three microservices example.	151

Acknowledgements

Chapters 4 and 5 contain material previously published in Van Eeden, Paier, and Miculan (2024), Paier, Van Eeden, and Miculan (2024), and Paier, Van Eeden, and Miculan (2026).

Chapter 8 contains material previously published in Baldo, Ion, Miculan, Paier, and Riccio (2025).

The use of generative AI was limited to grammar checking and language refinement; all content was written and fully reviewed by the author.

Vita

- 19 November 1997** Born, Vittorio Veneto (Treviso), Italy
- 2016–2019** Bachelor’s Degree in *Computer Science*
Final mark: 110/110 cum laude
University of Udine, Italy
- 2019–2022** Master’s Double Degree in *Computer Science*
Final mark: 110/110 cum laude
University of Udine, Italy
University of Klagenfurt, Austria
- May–Oct. 2022** *Research Fellow*
Department of Mathematics,
Computer Science and Physics
University of Udine, Italy
- Dec. 2025–Jul. 2026** *Research Grant*
Department of Legal Sciences
University of Udine, Italy

Publications

1. Baldo, Massimiliano, Pietro Di Gianantonio, **Matteo Paier**, and Marino Miculan (2026). “Strobilus: Enriching Cedar with Stateful Policies”. In: *Proceedings of the 31st ACM Symposium on Access Control Models and Technologies. SACMAT '26*. Canada: Association for Computing Machinery, pp. 205–216. DOI: 10.1145/3750555.3811892.
2. Baldo, Massimiliano, Marino Miculan, and **Matteo Paier** (2026). “Policy Automata for Stateful Authorization”. In: *Proceedings of the 27th Italian Conference on Theoretical Computer Science, ICTCS 2026, Udine, Italy, September 7-9, 2026*. To appear. CEUR-WS.org.
3. Miculan, Marino, **Matteo Paier**, and Jacopo Plozner (2026). “Experimental Evaluation of Lightweight Encryption Algorithms on 16-bit Microcontrollers”. In: *Proceedings of the 23rd International Conference on Security and Cryptography, SECRYPT 2026, Porto, Portugal, July 16-18, 2026*. SCITEPRESS, pp. 228–239. DOI: 10.5220/0015189200004103.
4. **Paier, Matteo**, Roberto Van Eeden, and Marino Miculan (2026). “Formal modelling and verifying eIDAS multi-factor authentication with interface-based threat analysis”. In: *Software and Systems Modeling*. DOI: 10.1007/s10270-026-01375-9.
5. Baldo, Massimiliano, Fabio Ionut Ion, Marino Miculan, **Matteo Paier**, and Vincenzo Riccio (2025). “OWSM: Empowering Rego for Stateful Access Control”. In: *Proceedings of the Joint National Conference on Cybersecurity (ITASEC & SERICS 2025), Bologna, Italy, February 03-08, 2025*. Vol. 3962. CEUR Workshop Proceedings. CEUR-WS.org. URL: <https://ceur-ws.org/Vol-3962/paper49.pdf>.
6. **Paier, Matteo**, Mattia Pizzolitto, Gian Luca Foresti, and Marino Miculan (2024). “Netstaldi: A Modular Distributed Architecture for Incremental Network Discovery”. In: *Proceedings of the 8th Italian Conference on Cyber Security (ITASEC 2024), Salerno, Italy, April 8-12, 2024*. Vol. 3731. CEUR Workshop Proceedings. CEUR-WS.org. URL: <https://ceur-ws.org/Vol-3731/paper37.pdf>.
7. **Paier, Matteo**, Roberto Van Eeden, and Marino Miculan (2024). “Formal Analysis of Multi-Factor Authentication Schemes in Digital Identity Cards”. In: *Software Engineering and Formal Methods - 22nd International Conference, SEFM 2024, Aveiro, Portugal, November 6-8, 2024, Proceedings*. Vol. 15280. Lecture Notes in Computer Science. Springer, pp. 423–440. DOI: 10.1007/978-3-031-77382-2_24.

8. Talotti, Enrico, **Matteo Paier**, and Marino Miculan (2024). “ECC’s Achilles’ Heel: Unveiling Weak Keys in Standardized Curves”. In: *Proceedings of the 8th Italian Conference on Cyber Security (ITASEC 2024)*, Salerno, Italy, April 8-12, 2024. Vol. 3731. CEUR Workshop Proceedings. CEUR-WS.org. URL: <https://ceur-ws.org/Vol-3731/paper09.pdf>.
9. Van Eeden, Roberto, **Matteo Paier**, and Marino Miculan (2024). “A Formal Analysis of CIE Level 2 Multi-Factor Authentication via SMS OTP”. In: *Proceedings of the 21st International Conference on Security and Cryptography, SECRYPT 2024*, Dijon, France, July 8-10, 2024. SCITEPRESS, pp. 483–491. DOI: 10.5220/0012768300003767.
10. Altarui, Andrea, Marino Miculan, and **Matteo Paier** (2023). “DBCChecker: a Bigraph-based Tool for Checking Security Properties of Container Compositions”. In: *Proceedings of the Italian Conference on Cyber Security (ITASEC 2023)*, Bari, Italy, May 2-5, 2023. Vol. 3488. CEUR Workshop Proceedings. CEUR-WS.org. URL: <https://ceur-ws.org/Vol-3488/paper01.pdf>.
11. De Nardin, Axel, Silvia Zottin, **Matteo Paier**, Gian Luca Foresti, Emanuela Colombi, and Claudio Piciarelli (2023). “Efficient few-shot learning for pixel-precise handwritten document layout analysis”. In: *IEEE/CVF Winter Conference on Applications of Computer Vision, WACV 2023*, Waikoloa, HI, USA, January 2-7, 2023. IEEE, pp. 3669–3677. DOI: 10.1109/WACV56688.2023.00367.
12. Miculan, Marino and **Matteo Paier** (2023). “Assembling Coherent Network Topologies Using Round-Trip Graphs (short paper)”. In: *Proceedings of the 24th Italian Conference on Theoretical Computer Science*, Palermo, Italy, September 13-15, 2023. Vol. 3587. CEUR Workshop Proceedings. CEUR-WS.org, pp. 110–115. URL: <https://ceur-ws.org/Vol-3587/6692.pdf>.
13. De Nardin, Axel, Silvia Zottin, **Matteo Paier**, Gian Luca Foresti, Emanuela Colombi, and Claudio Piciarelli (2022). “Dynamic Instance Generation for Few-Shot Handwritten Document Layout Segmentation (short paper)”. In: *Proceedings of the 1st Workshop on Artificial Intelligence for Cultural Heritage, AI4CH 2022*, co-located with the 21st International Conference of the Italian Association for Artificial Intelligence (AIIA 2022), Udine, Italy, November 28, 2022. Vol. 3286. CEUR Workshop Proceedings. CEUR-WS.org, pp. 26–34. URL: https://ceur-ws.org/Vol-3286/03%5C_paper.pdf.
14. Miculan, Marino and **Matteo Paier** (2022). “A Calculus for Subjective Communication”. In: *Proceedings of the 23rd Italian Conference on Theoretical Computer Science, ICTCS 2022*, Rome, Italy, September 7-9, 2022. Vol. 3284. CEUR Workshop Proceedings. CEUR-WS.org, pp. 148–160. URL: <https://ceur-ws.org/Vol-3284/1841.pdf>.

Presentations

1. “A Calculus for Subjective Communication”, 23th Italian Conference on Theoretical Computer Science (ICTCS 2022), 7–9 September 2022, Rome, Italy.
2. “DBCChecker: a Bigraph-based Tool for Checking Security Properties of Container Compositions”, Italian Conference on Cybersecurity (ITASEC 2023), 2–5 May 2023, Bari, Italy.
3. “Assembling coherent network topologies using round-trip graphs”, 24th Italian Conference on Theoretical Computer Science (ICTCS 2023), 13–15 September 2023, Palermo, Italy.
4. “A Formal Analysis of CIE Level 2 Multi-factor Authentication via SMS OTP”, Italian Conference on Cybersecurity (ITASEC 2024), 8–12 April 2024, Salerno, Italy.
5. “A Formal Analysis of CIE Level 2 Multi-Factor Authentication via SMS OTP”, 21st International Conference on Security and Cryptography (SECRYPT 2024), 8–10 July 2024, Dijon, France.
6. “Formal Analysis of Multi-Factor Authentication Schemes in Digital Identity Cards”, 22nd International Conference on Software Engineering and Formal Methods (SEFM 2024), 4–8 November 2024, Aveiro, Portugal.
Best paper award.
7. “How not to organise an onsite CTF”, Workshop on CTF Competitions & Offensive Security Training for Risk Awareness and Management, 12 December 2025, Bologna, Italy.
8. “The adoption of DLT against disinformation: showcasing TRUTHSTER Proof of Concept”, 8th Distributed Ledger Technology Workshop (DLT 2026), 4–6 June 2026, Pula, Italy.
9. “Experimental Evaluation of Lightweight Encryption Algorithms on 16-bit Microcontrollers”, 23rd International Conference on Security and Cryptography (SECRYPT 2026), 16–18 July 2026, Porto, Portugal.

Abstract

The rapid evolution of distributed ecosystems, ranging from web-based authentication platforms to industrial IoT and autonomous robotic swarms, has introduced unprecedented security challenges. As these systems become more heterogeneous, integrating diverse hardware, software stacks, and communication paradigms, traditional “all-or-nothing” security assessments often fail to capture the nuances of modern, multi-layered threats. This thesis addresses the problem of *automatically verifying security properties* in complex environments where trust assumptions are *partially* violated.

The core contribution of this work is the development of a unified formal framework for the symbolic verification of security protocols across both coupled (point-to-point) and decoupled (publish/subscribe) communication architectures. By shifting the focus from monolithic entities to an interface-based modelling approach, we enable a fine-grained threat analysis. This allows for the systematic exploration of “what-if” scenarios, where an adversary might compromise specific functional interfaces, while other components remain secure.

We first apply this methodology to the Italian Electronic Identity Card (CIE) ecosystem, identifying and mitigating vulnerabilities in multi-factor authentication schemes through dynamic attack scenario exploration. The framework is then extended to uncoupled systems by introducing a novel compositional model based on *directed hypergraphs*. This model formalises unique security dynamics, such as *read* and *write amplification* laws inherent in topic-based communication.

To address the limitations of stateless security measures, the thesis further investigates the integration of *service mesh architectures* as enforcement points for end-to-end security invariants. We propose and implement stateful access control policies within the mesh, with a dedicated management layer to handle state-aware requirements.

The effectiveness of the proposed approach is demonstrated through the verification of the Atomic Broadcast protocol, showing how the combination of formal modelling and service mesh orchestration can successfully neutralise complex attack scenarios that bypass traditional defences. Ultimately, this research provides a rigorous, platform-independent benchmark for securing modern, heterogeneous distributed systems.

Keywords: Symbolic formal verification, Fine-grained threat modelling, Heterogeneous systems, “What-if” analysis, Service mesh, Stateful access control.

Chapter 1

Introduction

The digital era has witnessed an exponential growth in the complexity of distributed systems, evolving from basic client-server models into intricate, decentralised ecosystems. As these networks become the foundation of critical infrastructure, spanning traditional web services and authentication platforms to industrial IoT and autonomous robotics, the demand for rigorous, provable security guarantees has reached an unprecedented level. In these environments, a single point of failure or a solitary logic error can cascade across the system, leading to systemic collapses.

Furthermore, the security landscape has grown increasingly sophisticated, with highly capable adversaries now launching multi-layered attacks. Modern security frameworks must address a wide range of vulnerabilities across both software and hardware.

At the infrastructure level, compromises in the computing platform or software, such as malware and exploits targeting the operating system or container orchestration layers, can enable attackers to completely bypass security guarantees. Moreover, misconfigurations in cloud permissions and policies remain a primary driver of data breaches, frequently resulting in the granting of excessive privileges to unauthorised users.

Once an initial foothold is gained, Advanced Persistent Threats (APTs) often employ lateral movement to navigate decentralised networks in search of high-value targets. In cyber-physical domains like robotics and

industrial IoT, these threats even extend to the physical layer: attackers may manipulate environmental conditions or inject malicious data directly into sensors to manipulate the system's underlying logic.

To combat these evolving risks, security cannot be an afterthought. It requires a thorough investigation into how protocols are defined, executed, and verified across every layer of the modern technological stack.

Consider the domain of Autonomous Vehicles (AVs), particularly scenarios involving heterogeneous swarms of drones, autonomous surface vessels, and underwater Remotely Operated Vehicles (ROVs).

To illustrate the security challenges in such a system, imagine a high-stakes mission designed to locate and report unexploded ordnance at the bottom of a lake. By working together, AVs can efficiently search large areas, share information, combine sensor data, and ensure redundancy in case of malfunctions. Communication is essential to coordinate different actors and make them cooperate to enhance safety, improve accuracy, and achieve their objectives more effectively. For example, since the GPS system cannot be used underwater, just to triangulate the position of a single submersible there is a need for multiple floating vessels with synchronised equipment aboard. The security challenges in this context encompass all aspects of the CIA triad.

Integrity is a priority, as the modification of messages could lead to the system overlooking real threats, endangering the public. For instance, if a malicious actor were to alter the coordinates of an unexploded ordnance, it could lead to search teams being directed to the wrong location, wasting valuable resources and potentially endangering lives. *Availability* is also critical, as delays in research operations could pose risks to both operators and the environment. For example, if the communication systems of the AVs were compromised, it could disrupt their ability to coordinate. Also, a similar AV scenario could be used for time-sensitive Search and Rescue operations, so this property is most definitely interesting to be considered. Finally, also *confidentiality* is essential to prevent the disclosure of sensitive information about the explosives and their locations. Such information could be used by malicious actors to plan attacks or to steal the explosive before their proper disposal.

To address these challenges, often the industry has proposed leveraging the Robot Operating System 2 (ROS 2025) framework to manage communication both within individual vehicles (between components) and between different robots. This approach offers a robust and flexible solution for ensuring the safety, security and reliability of AV operations. Notice that, despite the name, ROS is not a proper operating system, but rather an open source toolbox for building robot applications.

ROS provides a standardised framework for communication and co-ordination between different components of a robotic system. By using ROS, one can implement various security measures, such as encryption, authentication, and access control, to protect against unauthorised access and data tampering. Additionally, while outside our immediate scope, ROS telemetry modules can monitor AV behaviour to detect anomalies: by identifying deviations from expected operational patterns, these can flag potential security breaches in real time.

One core concept of ROS is its *plumbing* system, which handles communication between different modules (*nodes*) of robots. This messaging system employs a publish/subscribe pattern and relies on the Data Distribution Service (DDS Foundation 2025) standard. DDS is a middleware standard for data-centric communication. It enables efficient data sharing by creating a virtual global data space from which nodes can retrieve remote data transparently. Other key features include dynamic discovery of publishers and subscribers, and the scalable-by-design architecture. DDS is particularly well-suited for real-time applications like autonomous vehicles and IoT devices. It is widely adopted across various industries, including transportation, medical, aerospace, and military sectors.

Of particular relevance to our research, the DDS incorporates security mechanisms designed to safeguard data transmission across the system. These mechanisms are implemented via a pluggable architecture that includes authentication, access control, confidentiality, and message integrity. These security measures are designed to be enacted without compromising the strict real-time performance requirements of autonomous systems: DDS ensures that mission-critical data remains secure without introducing jitter or unacceptable latency.

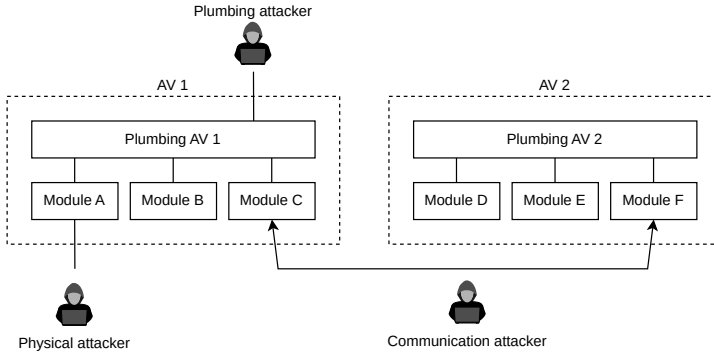


Figure 1.1: A ROS system with two AVs. The different levels of attacker interference are shown.

A significant threat vector involves the potential manipulation of ROS interfaces. An attacker could exploit architectural vulnerabilities or protocol weaknesses to intercept and modify the data flow between internal ROS modules. As illustrated in Figure 1.1, this potential interference manifests across three levels:

Physical an attacker could directly manipulate the raw data collected by sensors, either through attacks on the physical medium (e.g. GPS spoofing) or by tampering with sensor hardware (e.g. supply chain attacks). For instance, an attacker could use GPS spoofing techniques to send false location data to an AV, causing it to deviate from its intended path and collide with obstacles or other vehicles;

Plumbing the attacker could eavesdrop on or inject data into the internal vehicle communication bus, exploiting vulnerabilities in legitimate modules or introducing malicious modules. This could allow an attacker to gain control of critical functions of the AV, such as its steering, braking, or acceleration systems;

Communication the attacker could influence the communication between different AVs. For example, an attacker could intercept and modify messages sent between AVs, causing them to make incorrect decisions or coordinate their actions incorrectly.

Numerous attack scenarios can arise from various combinations of these capabilities. For example, an attacker might employ GPS spoofing, passively monitor internal AV communication (sniffing), and refrain from interfering with inter-AV communication. All the combinations of these capabilities lead to a vast number of different attack scenarios.

1.1 Problem statement

A fundamental challenge in modern systems engineering is the automatic verification of heterogeneous systems, where diverse hardware components, software stacks, and communication protocols must converge into a cohesive unit. In these environments, one of the primary concerns is not merely whether a system works under ideal conditions, but rather how it behaves when the underlying assumptions of trust are violated.

We must systematically investigate the “what-if” scenarios: what occurs when specific components begin to misbehave? Such deviations from expected behaviour may stem from a spectrum of sources, ranging from accidental misconfigurations and software regressions to intentional malicious exploits, malware, or even byzantine faults where components act in an arbitrary or contradictory manner.

A significant complication in this verification process is the fact that we need to model the attacker at different granularity levels. Unlike traditional “all-or-nothing” security assessments, modern threats operate at varying levels of intensity and access. An adversary’s control may be restricted to read-only eavesdropping on sensitive telemetry, or it may extend to read-write capabilities that allow for the injection of forged commands. Furthermore, this control is rarely uniform across the entire system: an attacker might compromise only specific components, or even isolated functions within a single module.

Modelling such a fragmented threat landscape makes formal verification of protocols more difficult. Traditional verification tools often struggle to represent an attacker who possesses partial levels of control across a distributed topology, leading to a difference between theoretical safety proofs and real-world vulnerabilities.

This difficulty is further increased by the shift toward uncoupled communication paradigms, such as multicast or publish-subscribe, which is central to modern distributed systems. While the Dolev-Yao model has long been the standard for verifying cryptographic protocols, it was primarily designed for coupled, point-to-point communication. It lacks a clear, standardised formalisation for the complexities of multicast and broadcast dynamics inherent in uncoupled systems. For example, in pub/sub, the decoupled nature of publishers and subscribers creates a wider attack surface where the traditional boundaries of a channel are blurred, complicating the application of automated formal analysis tools.

Furthermore, a recurring theme in the engineering of robotics and industrial IoT is that security is often treated as an afterthought. In these domains, real-time requirements (low latency, high throughput, and strict jitter bounds) are frequently prioritised over confidentiality or integrity. Designers often delegate security entirely to the transmission medium or the underlying network layer, assuming that a secure pipe is sufficient to protect the application logic. For instance, the industry-standard approach for securing MQTT involves leveraging TLS as the transport level protocol. However, this reliance on the medium ignores the threats mentioned earlier: if a component within the secure perimeter is compromised or misconfigured, the underlying secure transport will faithfully deliver its malicious intent. To achieve true resilience, we must move beyond perimetric security and integrate verifiable authorisation logic directly into the system components.

Finally, beyond basic confidentiality and integrity, robust security frameworks for distributed communication also may enforce temporal properties and fine-grained access control over shared resources and components. In these environments, authorisation is rarely static: rather, it must be considered inherently stateful. Classical models, such as Bell-LaPadula, demonstrate that access decisions often depend on the current security state and the classification levels of interacting entities. To rigorously secure systems, we require a framework capable of formalising these dynamic interactions, ensuring that state transitions remain secure over time and that temporal constraints are maintained.

1.2 Research questions

To establish a solid framework for the verification of heterogeneous autonomous systems, several critical research challenges must be addressed. These range from the formalisation of traditional coupled communication under fine-grained threat models to the extension of these models to uncoupled, asynchronous systems. Furthermore, a refined model must account for the presence of intermediary proxies, such as those found in service mesh architectures, to ensure end-to-end security invariants. Ultimately, this necessitates the integration of stateful access control within these meshes, leveraging specialised policy languages to handle the context-aware requirements of distributed systems access control.

RQ1 *How can we construct formal models for point-to-point (coupled) protocols that account for an adversary with fine-grained, interface-level compromise?*

This addresses the limitations of traditional “all-or-nothing” security models, which typically assume a binary state where a node is either fully trusted or entirely compromised. In complex systems, an adversary may instead gain fine-grained, interface-level control, such as read-only access to a specific telemetry stream or the ability to inject commands into a single actuator interface while other modules remain secure. By constructing a formal model that accounts for these varying degrees of compromise, including read-only, read-write, or partial component subversion, we can more accurately verify the resilience of a system against lateral movement and localised exploits. This shift allows for a more realistic assessment of how a single compromised interface impacts the end-to-end security invariants of the entire protocol.

RQ2 *How can the aforementioned models be extended to accommodate decoupled communication patterns (e.g. publish-subscribe) while maintaining the computational efficiency required for automated formal verification?*

This bridges the gap between traditional formal verification tools, which are primarily optimised for synchronous protocols, and the decoupled nature of modern communication paradigms. For instance, in a

publish-subscribe architecture, such as DDS, the lack of a direct connection between many-to-many nodes introduces significant complexity for automated analysis. The challenge lies in extending the attacker model to account for these interactions without creating a state space explosion. By identifying the minimal set of formalisms required to represent topic-based communication, this question seeks to ensure that the resulting model remains computationally tractable, allowing automated solvers to verify system-wide security properties efficiently.

RQ3 *How can we formally represent the internal logic and sidecar proxy architectures of service meshes within our protocol verification framework?*

This examines the transition from traditional end-to-end security to intermediary-based enforcement. In modern architectures, the sidecar proxy acts as a gatekeeper that intercepts and manages all inbound and outbound traffic for a service. Formally representing this architecture is essential because part of the security logic, such as authentication or login attempts, no longer resides within the application code itself, but within the service mesh layer. By incorporating the internal logic of these proxies into our protocol verification framework, we can analyse whether the middleware introduces new failure modes or successfully preserves the system's security invariants. This allows us to treat the service mesh not merely as a communication channel, but as an active, verifiable participant in the protocol that can contribute to end-to-end safety and security.

RQ4 *How can specialised policy languages be integrated into service mesh architectures to enable the enforcement of stateful access control, and what formalisms are required to verify their correctness during dynamic state transitions?*

This explores the integration of specialised policy languages into the service mesh layer to move beyond static permissions. The challenge lies in defining how these stateful policies are able to remain correct and consistent during dynamic state transitions. Enforcing this requires the policy engine to maintain a persistent internal state that is able to evolve during subsequent requests. By linking the service mesh's interception capabilities with a state-aware policy evaluator, we can ensure that per-

missions are dynamically revoked or granted as the system transitions between different phases, preventing unauthorised actions that might be valid in one context but dangerous in another.

1.3 Contributions

This thesis provides three main contributions to the field of formal verification of distributed systems security, spanning theoretical modelling, actual vulnerability discovery, and architectures for stateful authorisation.

Unified formal model for heterogeneous communication. The first major contribution is the development of a unified formal model designed to analyse both coupled (point-to-point) and uncoupled (multicast or publish-subscribe) communication within a single framework (Chapters 4 and 6). This is important since modern systems are inherently hybrid. For example, a robot may use a point-to-point TLS connection for a firmware update while simultaneously broadcasting telemetry over a decoupled DDS topic. By providing a common ground, our model enables the verification of protocols where these two paradigms coexist and interact. This is an important step for securing complex, modern scenarios where a compromise in a decoupled stream might be used to pivot and attack a coupled control channel.

To implement this contribution we follow a multi-stage methodology providing a path from high level system modelling to lower level formal proofs: its flowchart is available in Figure 1.2.

The process begins with the structural decomposition of the system components into their constituent interfaces. Rather than treating a heterogeneous system as a black box, we map every point of interaction, distinguishing between synchronous point-to-point links and decoupled channels. For each component, we explicitly specify the protocols to be analysed, defining the expected message types, cryptographic primitives, and state transitions (Chapter 4).

Once the protocols are defined, we enumerate the scenarios that must be tested. This involves defining different configurations of the attacker

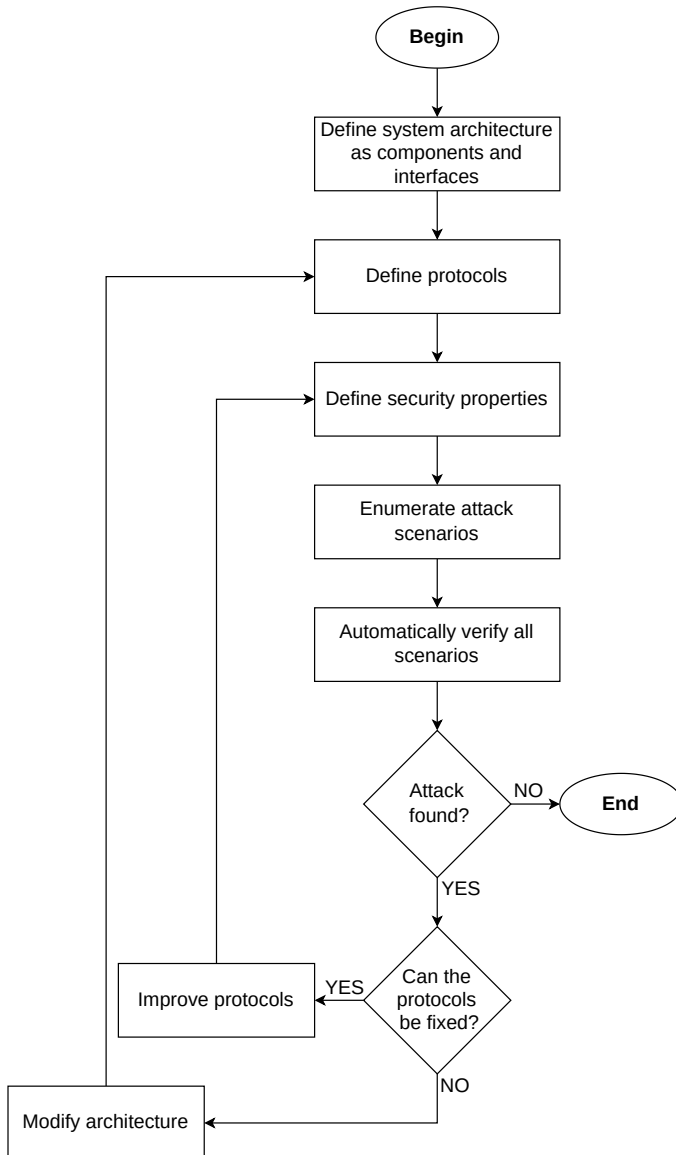


Figure 1.2: Flowchart for the proposed methodology. The two different loops, one for when fixing protocols, and one for when changing the architecture itself, are clearly visible.

model, varying the adversary’s control over specific interfaces (e.g. an attacker who can read a certain topic but cannot write to a coupled interface). Given the complexity of modern distributed systems, and our objective of providing a detailed description of fine-grained attacker capabilities, the analysis frequently triggers a state-space explosion, where the combinatorial growth of possible attack permutations surpasses the capacity for manual inspection. To overcome this issue, in Chapter 5 we propose an automated generation framework that translates the architectural specifications into a structured language compatible with automated protocol verifiers, minimizing human error during the formalisation process.

The primary tool utilised for the automatic verification of these models is ProVerif, chosen for its efficiency in analysing cryptographic protocols under the Dolev-Yao model. However, the methodology is designed to be tool-agnostic. Depending on the specific properties required several tools can be used: for example, for protocols requiring complex state or temporal properties, one can employ the Tamarin Prover (Meier et al. 2013). Additionally, the methodology supports a transition to fundamentally different verification paradigms, such as NuSMV or nuXmv for evaluating complex temporal logic or Rocq (formerly Coq) for constructing interactive mathematical proofs of correctness.

When an attack is successfully discovered, our methodology follows a two-tiered approach to remediation. Firstly, one can modify the protocol logic itself. For example, by adding nonces, timestamps, or additional signatures, it is possible to render the interaction inherently secure against the discovered exploit. On the other hand, in cases where the protocol cannot be easily changed (e.g. legacy systems), one can modify the system architecture. Nowadays, the typical architectural pattern for hardening the infrastructure is that of *service mesh*, where each service is guarded by a “sidecar proxy” whose function is to isolate compromised interfaces or enforce stricter access control policies that neutralise the attacker’s capabilities (Chapter 7).

Empirical validation: securing the Italian eID (CIE). To demonstrate the practical utility and expressivity of the proposed model, we applied it

to a high-stakes, real-world target: the Italian eID card (*Carta d'Identità Elettronica*, or CIE), which is an eIDAS-compliant identity system. Through this formal analysis, we successfully identified and proposed patches for several vulnerabilities within the protocol implementation (Chapter 5).

This contribution serves as a test for the model, proving that it is not merely a theoretical exercise but it can become a usable tool capable of uncovering subtle flaws in security infrastructures. The discovery and subsequent mitigation of these vulnerabilities highlight the model's ability to capture the intricacies of authentication protocols and its readiness for industrial application.

Stateful policy enforcement in service meshes. The final contribution addresses the statefulness gap in modern policy engines. In Chapter 8 we propose a novel architecture for integrating stateful policies into service meshes by wrapping existing stateless policy engines.

Most authorisation engines are snapshot-based, meaning that they make decisions based only on the data provided at the exact moment of the request. Moreover, they cannot directly interact or perform modifications to the data. Our approach introduces a management layer that tracks the system's state, effectively manipulating the output of stateless engines to enforce dynamic rules (e.g. “deny access to the actuator for the following 10 requests if the emergency sensor reads above a threshold”).

1.4 Structure of this thesis

The remainder of this thesis is structured to guide the reader through the theoretical foundations, practical applications, and architectural innovations developed to secure modern distributed systems.

Chapter 2 provides a comprehensive analysis of the state of the art, examining the current landscape of formal verification and identifying the limitations of existing security models when applied to heterogeneous systems. To equip the reader with the necessary technical background, Chapter 3 introduces the ProVerif verification tool, detailing the formalisms and applied π -calculus logic used throughout this work.

Building on these foundations, Chapter 4 introduces the initial system model and architectural framework specifically designed for coupled, point-to-point interactions. This model is put into practice in Chapter 5, where we formally model and verify the Level 2 and Level 3 protocols of the Italian Electronic Identity Card (CIE); here we also present the improvements and patches developed to address the specific vulnerabilities identified in the CIE protocols during our automated verification process.

The focus then shifts toward more complex environments in Chapter 6, which refines the initial model into a hybrid framework capable of handling both coupled and decoupled communication paradigms simultaneously. This chapter also includes the formal verification of a representative hybrid protocol to demonstrate the model's expressivity. Chapter 7 introduces the service mesh as a strategic architectural layer to further strengthen these protocols, verifying how sidecar proxies can enforce security invariants in hybrid environments.

In Chapter 8, we present OWSM, a system designed to wrap standard stateless policy engines to enable the distributed evaluation of stateful policies. This chapter details how the system manipulates engine outputs to maintain context-aware security across dynamic state changes.

Finally, Chapter 9 draws the thesis to a close by synthesizing the research findings and offering concluding remarks on the future of formally verified hybrid systems.

Chapter 2

State of the art

This chapter presents the state of the art by tracing the evolution of formal methods and infrastructure security, moving from foundational protocol verification to the architectural challenges of modern cloud-native environments. We examine the progression from the rigorous analysis of traditional cryptographic standards to contemporary interface-based and event-driven modelling, ultimately positioning these theories within the operational context of service meshes. This synthesis provides the theoretical and practical justification for the models developed in the subsequent chapters of this thesis.

2.1 Verification of protocols

Historically, Gritzalis et al. (1999) provides a foundational overview of the role of formal methods in the design and verification of cryptographic protocols within distributed systems. The authors define a protocol as a structured set of rules governing communication between agents and emphasise that the insecurity of open networks necessitates the use of cryptography to ensure confidentiality, integrity, and authentication.

The work categorises the application of formal methods into three distinct methodological branches. Firstly, *inference-construction methods* are introduced: these utilise specialised modal logics to model the evo-

lution of knowledge and belief among protocol participants (Burrows et al. 1990; Syverson et al. 1994). While widely adopted, Gritzalis et al. note significant challenges regarding the modelling of freshness, the idealisation of messages, and the lack of complete semantics for such logics. Secondly, there are *attack-construction methods*: these focus on the algebraic properties of protocol algorithms to identify potential attack sets (Dolev et al. 1983; Meadows 1992; Mitchell et al. 1997; Varadharajan 1989). Although independent of specific logic systems, these methods are frequently hampered by the state-space explosion problem, where the number of possible event permutations becomes computationally overwhelming. Finally, *proof-construction methods* emerge as a middle ground: this approach replaces exponential searches with theorems about those searches (Bolignano 1996). By modelling actual computations at a finer level of precision, proof-construction allows for thorough verification while maintaining computational tractability.

Furthermore, Gritzalis et al. distinguish between the *ex-post analysis approach* (verifying protocols after they have been designed) and the *ex-ante synthesis approach* (integrating formal methods into the initial design phase to prevent flaws by construction). This distinction is particularly relevant to modern distributed architectures, as it highlights the transition from simply detecting protocol failures to developing robust and error-free communication frameworks from the outset.

More recently, Basin et al. (2018) offer a comprehensive overview of the application of model checking to the security protocol domain, emphasizing its necessity for analysing interleaved protocol runs that are often too complex for manual inspection. The authors trace the evolution of the field from the seminal *Dolev-Yao model* (Dolev and Yao 1983), which abstracts cryptography as perfect and represents messages as symbolic terms. This abstraction serves as the foundation for modern verification, allowing tools to treat cryptographic operators as black boxes that only leak information if the corresponding key is known to the adversary.

The authors categorise the historical development of verification tools into two primary approaches. Firstly, *bounded session model checkers* rely on the result that the secrecy problem is NP-complete when the number

of sessions is restricted, enabling tools like OFMC and SAT-MC to search a finite state space. These are deemed highly effective for catching the majority of practical attacks, which typically require only a few interleaved sessions. Secondly, *unbounded session model checkers* encompass tools such as *ProVerif* (Blanchet 2008), utilizing abstraction to overcome the inherent undecidability of the infinite-session secrecy problem. By employing heuristics and symbolic approximations, these tools can provide proofs of security that hold for an arbitrary number of concurrent executions.

The work also highlights the successful transition of model checking from academic exercises to the standardisation of real-world protocols. The authors cite instances where tools like the NRL Protocol analyser and *ProVerif* were instrumental in verifying IETF standards, smart card protocols, and the TLS protocol (Bhargavan et al. 2008).

Significantly for this thesis, Basin et al. identify future challenges that include increasing the expressivity of the Dolev-Yao model to handle algebraic properties (such as associativity-commutativity) and extending formal reasoning to handle local state compromise. This aligns with our research goals of modelling fine-grained attacker control, effectively moving the field toward more nuanced protocol analysis.

Costa et al. (2025) provide a systematic evaluation of the impact of *Process Algebras* (PAs) on cybersecurity, emphasizing their role as the mathematical foundation for protocol verification. The authors argue that the success of PAs originates from their ability to offer a compact yet expressive syntax that captures essential computational behaviours while abstracting away irrelevant implementation details. This survey categorises the influence of PAs across three primary cybersecurity sub-fields: secure development, attack modelling, and vulnerability assessment.

The work is particularly significant for its mapping of specialised features introduced into PAs to handle specific security challenges. The authors identify several critical variants of process algebras that have evolved beyond the original CCS or π -calculus, such as *Security PAs*, which are explicitly designed to model threats and cryptographic interactions, and *Imperative PAs*, which embed assignment operators to update memory state during execution.

Furthermore, the inclusion of *Wireless* and *Cyber-Physical PAs* highlights the necessity of combining synchronous and asynchronous communication features when reasoning about hybrid physical-digital systems.

By tracing the chronological development of these formalisms, Costa et al. demonstrate how PAs have shaped the research community's approach to security. For the purposes of this thesis, their analysis of Cyber-Physical PAs provides a strong theoretical justification for the integration of stateful logic into formal frameworks. The survey concludes by highlighting that while PAs have been profoundly influential, certain application areas still remain underexplored.

Not strictly linked to this thesis, but still relevant, L. Li et al. (2017) address the critical challenge of verifying protocols that rely on *temporal constraints*. The authors argue that while time is a common security mechanism in modern systems, it is notoriously difficult to use correctly due to a historical lack of formal specification techniques that can account for continuous time and symbolic timing parameters. They introduce a comprehensive analysis framework based on a timed extension of the applied π -calculus, which allows for the natural modelling of both cryptographic operations and time-sensitive measurements.

The verification process is designed to be exhaustive, either producing a mathematical constraint on the timing parameters that guarantees security or identifying an attack that persists regardless of the parameter values. The utility of this framework was demonstrated by the discovery of a previously unknown timing attack in the Kerberos V protocol.

Finally, Balliu et al. (2021) investigate the security and safety risks inherent in *IoT automation platforms* like IFTTT, where third-party apps triggered by external sources to actuate physical or digital sinks can interact in unintended or malicious ways. The authors show that, while specific cross-app interferences have been documented, the field lacks a foundational semantic framework to understand these interactions and define sound security policies. This is particularly problematic in environments where apps affect a shared physical state, such as room temperature or window status, creating hidden dependencies between seemingly unrelated automations.

The core contribution of this work is a formal calculus that models the behavioural semantics of concurrently executing IoT apps. This framework generalises syntactic enforcement mechanisms by connecting them to *bisimulation-based* notions of security, providing a rigorous baseline for soundness criteria.

2.2 Interface-based modelling

In the domain of component-based modelling, De Alfaro et al. (2001) represents one of the first works in the literature to provide a rigorous formalisation of *interfaces*, establishing a distinction between component models and interface models. For the authors, while a component model describes a system’s behaviour in an arbitrary environment to support compositional verification, an interface model focuses on the assumptions a component makes about its environment to facilitate compositional design and refinement. This distinction is important for effective abstraction: where a component description answers what a piece of hardware or software does, an interface description specifies how it can be used and how it interacts with its surroundings.

Jacomme et al. (2021) provide a comprehensive formalisation of *multi-factor authentication* (MFA) protocols by introducing a threat model that moves beyond the standard Dolev-Yao network attacker. The authors argue that classical protocol analysis, which assumes the attacker only controls the communication network, is insufficient for modern MFA schemes like Google 2-step and FIDO U2F. Instead, they formalise a model in the applied π -calculus that accounts for compromised passwords, TLS channel abstractions, and varying levels of platform compromise, such as malware with specific read or write access to device interfaces.

A significant contribution of this work is the inclusion of *human aspects* into the formal model: by acknowledging that human actors may omit security procedures, the model provides a more realistic assessment of protocol security. By systematically generating all combinations of threat scenarios and using ProVerif for automated analysis, the researchers identified critical weaknesses in widely used protocols.

For this thesis, the methodology of Jacomme et al. is highly relevant as it demonstrates how to extend the applied π -calculus to model complex, real-world trust assumptions and device states. Their work provides a precedent for our work, where the authors similarly account for the potential compromise of specific service interfaces.

Barbosa et al. (2021) conduct a modular provable security analysis of the FIDO2 ecosystem, which comprises the Web Authentication (WebAuthn) specification and the Client-to-Authenticator Protocol (CTAP2). This research is important because it moves beyond verifying isolated protocols to analysing the interaction between distinct components, the browser/operating system platform and the external hardware authenticator. By proposing formal security models for both specifications, the authors were able to mathematically confirm the authentication security of WebAuthn while identifying significant design flaws in CTAP2.

The analysis reveals that CTAP2 only provides security in a weak sense against realistic adversaries. To address these vulnerabilities, Barbosa et al. introduce sPACA, a generic protocol designed to replace CTAP2. They provide a formal proof that sPACA achieves stronger security guarantees and improved performance compared to the original standard.

In another field, Bartocci et al. (2022) propose a contract-based design methodology to manage the complexity of developing secure systems. This approach utilises an interface theory that formalises the relationship between assumptions and guarantees, i.e. the security promises the component must fulfil. The authors present an interface theory specifically designed for ensuring system-wide security properties.

The framework is significant for its support of both *stateless and stateful interfaces*, enabling a dual-path design process. It supports top-down steps, where system-wide requirements are decomposed into smaller refinements, and bottom-up steps, where individual components are assembled into a larger system. By providing a formal refinement relation and a composition operation, the theory ensures that security properties are preserved as components are integrated. This methodology is demonstrated through an automotive domain example, showing how local contracts can be composed to satisfy global information-flow policies.

2.3 Publish-subscribe modelling

Baresi et al. (2006) address the inherent difficulty of validating distributed applications built on top of *publish-subscribe* infrastructures, noting that traditional verification often requires oversimplifying the communication layer to remain computationally feasible. They argue that the decoupling provided by this paradigm creates a gap between component-level behaviour and global system properties. The authors propose a domain-specific approach that embeds asynchronous publish-subscribe primitives directly into the model checker's language. This allows for a fine-grained representation of the infrastructure, accounting for various delivery and reliability guarantees, without triggering state-space explosion. By treating operations like publish and subscribe as native modelling constructs, the framework enables the automated verification of application components atop a realistic communication fabric.

In a follow-up work, Baresi et al. (2007) extend this methodology to account for the temporal aspects of decoupled interactions, which they identify as crucial for real-world applications. They observe that the unpredictability of components joining or leaving the system, combined with varying message delivery orders, makes reasoning about global behaviour at design time very challenging. By enabling their domain-specific model checker to verify temporal properties, they explore the interaction between time and the diverse semantics of publish-subscribe systems. The authors express the importance of formally verifying the verification tool itself to provide a strong foundation for the results.

Concluding this research line, Baresi et al. (2010) introduce *Loupe*, a domain-specific model checker designed to address the unique verification challenges of publish-subscribe architectures.

The authors argue that while the decoupling of publish-subscribe systems facilitates the design of dynamic applications, it complicates reasoning about the overall federation of components. A primary obstacle in traditional verification is that existing publish-subscribe infrastructures offer radically different guarantees which affect the outcome of security and correctness analyses.

The technical core of Loupe involves embedding publish-subscribe primitives directly into the checking engine of the Bogor model checker (Robby et al. 2003), rather than modelling the infrastructure on top of a general-purpose language. This approach treats operations like publish and subscribe as atomic constructs of the modelling language. By doing so, Loupe avoids the state-space explosion typically caused by modelling the internal dispatcher logic as an explicit part of the system state.

Furthermore, the authors highlight that in publish-subscribe architectures, the critical information determining the system state resides primarily in the application components rather than in the communication infrastructure. Loupe keeps the dispatcher’s internals transparent to the application model, allowing for a high-fidelity representation of multipoint communication and binding between publishers and subscribers.

This work provides a precedent for treating *decoupled communication* as a first-class citizen in formal verification. The abstractions introduced in Loupe, specifically the handling of multipoint delivery and the separation of infrastructure logic from application state, justify our approach to modelling hybrid systems where both point-to-point and event-driven interactions must be verified in a computationally efficient manner.

Khelifi et al. (2008) examine the verification of publish-subscribe architectures as complex systems characterised by *emergent processes* and dynamic federations of components. They highlight that the decoupling provided by publish-subscribe middleware enables high evolvability but necessitates rigorous consistency checks during dynamic reconfigurations. To address this, the authors propose a modelling approach using automata with multiplicities, which generalises finite state automata by labelling transitions with dual values representing communication modes and the specific events deposited by entities.

For the validation phase, the authors use the SPIN model checker and its modelling language, PROMELA. This choice is driven by SPIN’s ability to model both synchronous and asynchronous communication through local channels, and its native support for *Linear Temporal Logic* (LTL), which allows for the formal specification of dynamic behavioural properties and the sequence of states during system execution.

By utilizing both assertions within the PROMELA code and external LTL formulas, the framework can verify structural and behavioural properties even as new components join the system. This methodology aligns with the broader research goal of maintaining architectural style and consistency in reconfigurable distributed systems.

On the topic of publish-subscribe broker security, Ion (2013) investigates the architectural mismatch between the host-centric design of the Internet and the rising demand for content-centric applications, identifying the publish-subscribe paradigm as the most mature solution for decoupling senders and receivers. The research highlights that while traditional point-to-point security mechanisms are insufficient for these decoupled environments, the core challenge lies in the role of the distributed brokers. In large-scale applications brokers often span multiple administrative domains and cannot be fully trusted with the plaintext content of the messages they route.

The primary contribution of this work is the formalisation of the *confidentiality-preserving routing* problem. The author argues that a secure publish-subscribe system must allow brokers to perform content-based matching against registered subscriptions without the brokers learning sensitive details of either publications or subscription filters. This task is particularly difficult when subscriptions involve complex logical conjunctions and disjunctions of numerical and non-numerical inequalities.

Similarly, Chien et al. (2025) address the inherent privacy risks in the standard MQTT publisher–broker–subscriber architecture, where brokers have the capacity to inspect message contents. While end-to-end encryption is often proposed as a solution to protect data from *curious brokers*, the authors identify a significant bottleneck: in traditional end-to-end models, messages are encrypted at the application layer for the subscriber and then re-encrypted at the transport layer for the broker, forcing the broker to decrypt and re-encrypt messages for every subscriber in a group.

The researchers propose and formally verify two efficiency-enhanced solutions: a group key-based approach and a client–broker channel integrity-only approach. By optimizing the protocol to eliminate redundant encryption layers while maintaining cryptographic integrity, they

significantly reduce the computational overhead on both the clients and the central broker. Their work includes formal security proofs to ensure that these performance gains do not compromise the original security goals of confidentiality and authenticity against adversarial peeking.

Zhou et al. (2016) explore the challenges of ensuring consistency during the dynamic evolution of service-oriented systems. They argue that for system evolution, such as updating service logic, to be trusted, the process must be formally verified across three distinct phases: before, during, and after the evolution occurs. To achieve this, the authors propose a verification framework based on *hierarchical timed automata*, which allows for the direct modelling of temporal constraints and the structural decomposition of complex software architectures.

A significant contribution of this work is the integration of probabilistic transitions to account for the uncertainty inherent in distributed environments. This enables the verification of parameter-level evolution, where system behaviour may be non-deterministic. The methodology involves a flattening algorithm that translates these high-level hierarchical models into sequential timed automata, making them compatible with mainstream model checkers like UPPAAL and its statistical extension, UPPAAL-SMC. By leveraging temporal logics to specify safety and liveness properties, the framework can mathematically guarantee that a system remains consistent with its specification even as its internal services undergo dynamic changes.

Finally, Delzanno (2019) presents a formal model for publish-subscribe architectures focused on the central communication broker’s role in message distribution. Unlike models that treat the broker as a transparent entity, this work explicitly considers the *internal structure* of the server, such as worker threads and global subscriber lists, to analyse how different notification semantics affect system behaviour. The author addresses protocols designed for an arbitrary number of components, a challenge that typically falls under the parameterised verification domain.

The paper’s methodology leverages Cubicle, an SMT-based infinite-state model checker, which is particularly suited for handling systems that combine asynchronous communication with synchronous operations

on local data structures. This allows for a more faithful representation of real-world publish-subscribe protocols like MQTT or Redis. The author explores the coverability decision problem to identify reachability of bad configurations regardless of the number of clients involved.

A key aspect of this research is the use of labelled automata to specify client behaviour, while the server is modelled as a transition system managing a finite set of resources. The model can detect vulnerabilities that only emerge under specific, potentially rare, initial system states.

On the topic of MQTT security, Aziz (2016) presents a formal analysis of the MQTT 3.1 protocol utilizing a timed message-passing process algebra. By constructing a formal model, the author is able to identify significant ambiguities within the original protocol specification, particularly regarding the third Quality of Service (QoS) level. To address these issues, they proposed enhancements to the protocol specification, again demonstrating the importance of formal methods in ensuring the reliability and safety properties of communication protocols in decoupled environments.

About more practical approaches to the security of *Data Distribution Service* (DDS) standard, Kim et al. (2021) suggests integrating *Attribute-Based Access Control* (ABAC) to enable more granular control based on message content, rather than focusing primarily on participant-based authorisation, like the current DDS security model usually does. This integration allows for more flexible and fine-grained authorisation policies. The proposed model was implemented and evaluated in a healthcare scenario, demonstrating that the framework is effective in enforcing access control in dynamic environments while remaining scalable and efficient.

White et al. (2017) explore the security challenges arising from the convergence of *Operational Technology* (OT) and *Information Technology* (IT) within the *Industrial Internet of Things* (IIoT). As critical infrastructure increasingly adopts IT-driven connectivity, the authors argue that the security of application-layer messaging protocols, traditionally overlooked in isolated industrial settings, has become a pressing and high-priority concern. Their research specifically targets the DDS Interoperability Real-Time Publish-Subscribe (DDSI-RTPS) wire protocol, identifying some problems in its security.

Through simulations in a virtual environment, the study demonstrates that the discovery mechanisms of DDSI-RTPS are insecure, allowing non-authenticated clients to perform device enumeration in both active and passive modes. Other than enumeration, the authors discover that the manipulation of sequence numbers can be used to enable denial-of-service attacks, and that the fabrication of malicious heartbeat messages can deny the receipt of legitimate data.

Finally, Liu et al. (2018) present a formal verification approach for DDS within the ROS2 framework using the *PRISM model checker*. Their findings demonstrate that the system can effectively prioritize high-priority messages and meet specific timing requirements. Additionally, the study suggests that system performance can be further optimised by increasing the message buffer size.

2.4 Service meshes

W. Li et al. (2019) provide a comprehensive survey of the service mesh as a critical evolution in software service delivery, emerging to manage the operational complexity inherent in microservice architectures. The authors trace the shift from monolithic, on-premises designs to cloud-native architectures where applications are composed of hundreds or thousands of loosely coupled, language-agnostic services. While container orchestration tools like Kubernetes have eased deployment, they have simultaneously increased runtime complexity due to the dynamic nature of service instances and their constantly changing states.

The paper defines the service mesh as a dedicated infrastructure layer that standardises service-to-service communication without requiring modifications to the application code. This separation of concerns is achieved by offloading communication-related tasks, such as traffic segmentation, interoperability, and runtime enforcement, to a specialised platform. W. Li et al. identify three primary challenges that the service mesh is meant to address:

Heterogeneity meaning managing services implemented in diverse languages and belonging to different tenants;

Dynamic environments i.e. handling the volatility of service instances caused by orchestration scheduling;

Traffic Control enforcing security and routing policies based on the flow of traffic between dependencies.

For the scope of this thesis, this work serves as a justification for using the service mesh as the enforcement point for our proposed models. W. Li et al. highlight that while the mesh provides the mechanism for runtime enforcement, the complexity of service dependencies makes it a challenging task to ensure overall system integrity. Thus arises the need for formal verification to prove that the communication-related concerns addressed by the mesh actually meet the intended security guarantees.

Chandramouli et al. (2021), writing for the United States National Institute of Standards and Technology (NIST), establishes the industrial and architectural standards for securing modern cloud-native applications. This work highlights the transition from monolithic architectures to loosely coupled microservices, where a dedicated infrastructure layer (the service mesh) is used to provide critical security services independently of the application code. This architectural shift is fundamental to implementing zero trust principles, ensuring that mutual authentication and fine-grained authorisation occur between every pair of services.

A key contribution of this NIST guidance is the formalisation of the *Policy Enforcement Point* (PEP) and *Policy Decision Point* (PDP) within the mesh. By leveraging sidecar proxies to intercept traffic, the service mesh can decouple security logic from the business logic. Moreover, the document discusses how the features of the mesh can be augmented by calling external authorisation systems. This work provides practical context for this thesis: while NIST provides the deployment guidance and architectural blueprint for these systems, it does not offer formal methods to verify the complex interactions that arise when meshes are deployed in hybrid scenarios. Our work on stateful policy engines addresses the NIST objective of building operational assurances by providing a framework to wrap the stateless PEP/PDP structures and enable the stateful policy evaluation necessary for the dynamic scenarios described in this guidance.

Finally, Pallewatta et al. (2024) examine the security challenges of microservice architectures within the context of IoT applications, specifically focusing on multi-domain edge-cloud environments. They argue that while the independently deployable nature of microservices enhances performance and scalability, it also introduces significant risks during data processing and transmission across distributed and dynamically composed components. To mitigate these risks, the authors propose an architectural framework that leverages Policy-as-Code (PaC) to enforce fine-grained security controls at the microservice level.

The framework is built around an intelligent *control plane* that dynamically configures cloud-native technologies, such as container orchestrators and service meshes, to enforce security policies during critical management operations like initial placement, scaling, and migration. By utilizing a stack of open-source tools, including Docker, Kubernetes, Istio, and Open Policy Agent (OPA), the researchers demonstrate how security policies can be integrated into the deployment pipeline.

2.5 Stateful policies

Orlovsky et al. (2007) address the transition from centralised server environments to globally distributed systems, highlighting the inherent difficulty of defining and enforcing security policies when the global order of events is not natively defined. The authors distinguish between *stateless policies*, such as traditional authentication, authorisation, and access control, and *stateful policies*, which require knowledge of the execution history to make an informed decision. They argue that stateless mechanisms are often too restrictive or insufficient for modern needs, such as preventing data leakage, where a send operation should only be denied if sensitive internal data has been previously accessed.

The paper builds on the theoretical foundation of *Execution Monitoring* (EM), which operates by monitoring system events and intervening before a policy violation occurs. While traditional EM models rely on a centralised enforcement engine, Orlovsky et al. contend that such an approach is unfeasible for large-scale distributed systems due to network

congestion, latency, and the risk of a single point of failure. To solve this, they propose a decentralised mechanism consisting of multiple local instances of smart sandboxes. These instances share security-relevant information efficiently to enforce global stateful policies without the need for a central bottleneck.

For the context of this thesis, this work is instrumental in justifying the shift toward distributed enforcement mechanisms. Orlovsky et al.'s concept of a decentralised EM that shares state information mirrors our architectural approach to the service mesh sidecar proxy as a local policy enforcement point that can share state.

Recently, Sappa (2025) addresses the limitations of traditional security frameworks in cloud-native environments, which often fail to account for the highly dynamic and multifaceted nature of modern data streams. The author argues that existing enforcement mechanisms typically overlook contextual workload behaviour, user roles, and environmental metadata, leading to significant security gaps and elevated false-positive rates. To resolve this, they propose a context-aware policy enforcement framework integrated directly into Kubernetes-native layers, specifically admission controllers and service meshes. The core contribution of this work lies in its empirical validation of how contextual signals, such as real-time workload performance and metadata, impact the precision of security decisions. Through the analysis of a hybrid dataset involving synthetic microservices and actual violation logs, the study demonstrates that including these signals improves detection accuracy by 41% while maintaining extremely low latency (under 8ms in 95% of cases). Furthermore, the framework effectively mitigates *context drift*, where security policies become obsolete as the system's operational environment evolves.

For this thesis, the work by Sappa serves as a recent and compelling justification for the necessity of dynamic policy enforcement. Their focus on integrating context into the service mesh layer aligns perfectly with our development of stateful policy engines.

Not directly related to this thesis, but nonetheless important, Alboqmi et al. (2024) identify the risks associated with static and manually configured access control policies in standard service meshes. They argue

that while mutual TLS provides a baseline for identity, it often grants an implicit access that can be exploited if a mutually authenticated service is compromised. To mitigate this, the authors propose SMAAC, an architecture that shifts the service mesh from static enforcement to a *Risk-Adaptive Access Control* model.

The SMAAC framework introduces three primary components into the service mesh control plane to manage this transition. First, the runtime trust evaluator assesses the trustworthiness of a service by analysing behavioural patterns and assigning a dynamic trust metric to each request initiator. Second, the threat intelligence sharing mechanism gathers and distributes telemetry data, enabling the mesh to recognise and share attack tactics across the distributed environment. Finally, the access policy generation component serves as the self-adaptive core of the system. When a service's trust metric falls below a defined threshold, this component automatically generates and pushes new access control policies. These policies are based on trust and service sensitivity, which is quantified by the potential impact on confidentiality, integrity, and availability.

The researchers validated SMAAC using the Istio service mesh and the μ Bench microservice suite, demonstrating that self-adaptation allows the mesh to modify behaviour at runtime in response to perceived threats. For the purposes of this thesis, the work by Alboqmi et al. provides a sophisticated example of the stateful, dynamic policies that require rigorous analysis.

Chapter 3

Introduction to ProVerif

We present an overview of ProVerif, a tool for the symbolic automated verification of security protocols. ProVerif operates within the Dolev-Yao model, which treats cryptographic primitives as perfect black boxes and assumes an adversary with total control over the network. For a more comprehensive technical foundation, we refer to Blanchet et al. (2016) and the extensive documentation available online.

In ProVerif, protocols are formally specified using a variant of the *applied π -calculus*, a process algebra for modelling the behaviour of concurrent processes as they exchange data over potentially insecure channels. Within this framework, messages and their associated cryptographic operations are represented as abstract algebraic terms. This abstraction allows us to define *term constructors*, i.e. functions used to build messages, such as `senc(message, key)` for symmetric encryption; and *equations*, i.e. rules that characterise the algebraic properties of these operations.

Internally, ProVerif models the system by translating the protocol and the security properties into a set of Horn clauses, with the security properties specified as logical goals. A specialised resolution algorithm is then executed on this representation to search for a derivation of the negation of each goal. If no derivation is found, the security property is verified. However, if a derivation is found, ProVerif extracts a concrete attack trace and presents it to the user.

To determine the validity of security invariants, ProVerif internally translates the protocol's process logic and the desired security properties into a set of Horn clauses. In this representation, security properties are framed as logical goals that the solver must satisfy. The tool then utilises a specialised resolution algorithm to exhaustively search for a derivation that achieves the negation of these goals, effectively attempting to prove that a violation is possible. If the algorithm proves that no such derivation exists within the infinite state space, the property is formally verified as secure. If a derivation is successful, ProVerif reconstructs and presents a concrete attack trace, providing the user with a step-by-step execution path demonstrating how the adversary can compromise the protocol.

3.1 Terms and processes

ProVerif allows the user to define a formal algebraic theory of terms by declaring a *signature* Σ . This signature consists of a finite collection of function symbols and a corresponding set of algebraic laws that govern how terms are reduced.

Starting from this signature Σ , and utilising countable sets of *names* (representing atomic values like keys or nonces) and *variables*, the structure of terms is defined according to the following grammar:

$$\begin{array}{ll}
 M, N ::= a, b, \dots & \text{names} \\
 & | x, y, \dots \quad \text{variables} \\
 & | f(M_1, \dots, M_n) \quad \text{constructor application, with } (f : n) \in \Sigma
 \end{array}$$

More complex datatypes are constructed by pairing function symbols with specific equational laws. For example, *pairs* are defined through a signature $\Sigma_p = \{pair : 2, \pi_1 : 1, \pi_2 : 1\}$ and the reduction rules

$$\pi_1(pair(M, N)) = M \quad \text{and} \quad \pi_2(pair(M, N)) = N$$

In this context, M and N are treated implicitly as universally quantified variables. Because the functions π_1 and π_2 serve to unpack (eliminate) a previously applied constructor, they are called *destructors*.

This paradigm of constructors and destructors allows for the modelling of diverse cryptographic primitives. For instance, a cryptographic hash function is typically represented as a standalone constructor $h : 1$, with neither destructors nor reduction rules, since hashes are designed to be one-way and collision-resistant in the symbolic model. This way, it is ensured the original input cannot be algebraically recovered.

In this framework, a *perfect symmetric encryption* scheme is formalised by the signature $\Sigma = \{\text{senc} : 2, \text{sdec} : 2\}$ and the defining equation:

$$\text{sdec}(K, \text{senc}(K, M)) = M$$

This property captures the core security invariant of symmetric cryptography: the original message M is recoverable if and only if the exact key K used during encryption is provided to the decryption function. If an adversary attempts to decrypt using an incorrect key K' , the term $\text{sdec}(K', \text{senc}(K, M))$ remains irreducible: it cannot be simplified to M .

We call this scheme as “perfect” because, within the ProVerif environment, the encrypted term $\text{senc}(K, M)$ functions as a black box: it behaves as an ideal primitive, meaning the tool has no mechanism to extract partial information from the ciphertext without the key. As we are working within a symbolic model, we deliberately abstract away all computational complexities. ProVerif does not account for bit-level details such as key entropy, message length, probability distributions, or the feasibility of brute-force attacks; instead, it focuses strictly on the logical and structural properties of the protocol.

The complete specification of the process language, including its type declarations, is detailed in the ProVerif reference manual. For convenience, we present the essential syntax of the process grammar, defined according to a given term signature and its associated algebraic equations, in Figure 3.1.

The null process 0 denotes a terminated state where no further actions occur. Communication is handled by the blocking input $\text{in}(c, x); P$, which pauses execution until a message M is received on channel c , at which point it continues as P with x substituted by M ($P[M/x]$). Conversely, the output $\text{out}(c, M); P$ sends message M on channel c before proceeding.

$P, Q ::= 0$	null process
$\text{in}(c, x); P$	message input
$\text{out}(c, M); P$	message output
$P Q$	parallel composition
$!P$	replication
$\text{new } n; P$	fresh name creation
$\text{if } M \text{ then } P \text{ else } Q$	conditional
$\text{let } M = N \text{ in } P$	local declaration with pattern matching
$\text{event}(e(M)); P$	event

Figure 3.1: Syntax of ProVerif processes.

Concurrent behaviour is modelled via parallel composition $P|Q$, allowing processes to run and interact over shared channels simultaneously. To account for multiple protocol sessions, replication $!P$ provides an unbounded supply of parallel process instances, while restriction $\text{new } n; P$ ensures the generation of unique, local values like nonces or keys. Logic is controlled through the standard conditional $\text{if } M \text{ then } P \text{ else } Q$ and the $\text{let } M = N \text{ in } P$ pattern-matching construct, the latter of which proceeds only if M successfully matches the pattern N . Finally, the non-blocking $\text{event}(e(M_1, \dots, M_n)); P$ serves as a marker to signal that a specific event has occurred. This mechanism is fundamental for verifying complex security properties such as temporal correspondence and data integrity.

3.2 Security property analysis

As mentioned above, ProVerif conducts its protocol analysis by translating protocol logic into Horn clauses, which are formulas in first-order logic represented as

$$F_1 \wedge \dots \wedge F_n \Rightarrow G_1 \vee \dots \vee G_m$$

where each F_i and G_j is a literal. ProVerif utilises several built-in literals to track adversary capabilities; most notably, the predicate $\text{attack}(M)$

indicates that the term M is within the attacker's knowledge base. Furthermore, users can define custom literals to track protocol progression, such as event `event` $e(M_1, \dots, M_n)$, which holds true once a process has reached and triggered a corresponding event during its execution.

As said, ProVerif's verification engine maps the entire system (including cryptographic primitives, adversary capabilities, and the protocol logic) into an exhaustive set of Horn clauses. Security properties are then formulated as queries that the tool seeks to resolve. A fundamental example is the query:

`query attacker( $M$ )`

This query tasks the tool with determining whether an adversary can derive the message M , be it a secret key or a private nonce. ProVerif attempts to prove this by searching for a logical deduction of the fact from the established Horn clauses and the defined algebraic theory. If a valid deduction is found, the tool extracts a concrete attack trace from the inference chain to demonstrate the breach. Conversely, if no derivation exists within the search space, the security property is formally verified, confirming that M remains confidential and inaccessible to the attacker.

Beyond confidentiality, the correspondence assertion is another important query type used to enforce causal dependencies and temporal ordering between execution steps, even when they occur across logically separate participants. These assertions are structured as universally quantified implications:

`query event( $e(M_1, \dots, M_n)$ )  $\Rightarrow$  event( $e'(N_1, \dots, N_k)$ ).`

For the assertion to hold, the occurrence of event e must strictly guarantee that event e' has preceded it. In the context of protocol security, this is frequently used to model basic authenticity: for instance, if a receiver triggers an event `event(MsgRcvd( $M$ ))` for a specific message M , the protocol is only secure if the sender has previously triggered `event(MsgSnt( $M$ ))`.

To verify these properties, ProVerif attempts to falsify the assertion by searching for a counterexample, essentially trying to prove the negation of the goal:

$\exists M_1, \dots, M_n, N_1, \dots, N_k. \text{event}(e(M_1, \dots, M_n)) \wedge \neg \text{event}(e'(N_1, \dots, N_k)).$

If the solver succeeds in this inference, it indicates that the system can reach a state where the consequent event occurs without the antecedent having ever taken place. ProVerif then provides a concrete attack trace illustrating how an adversary can manipulate the system into this inconsistent state, thereby proving a violation of the intended security property.

Because the formal verification of security properties is an undecidable problem, ProVerif, along with comparable protocol verifiers, cannot guarantee a definitive conclusion for every model. Additionally, the analysis of highly intricate protocols may be restricted by hardware-specific time and memory constraints. In practice, the tool yields one of four outcomes:

True if the property is formally verified, confirming that no valid attack exists within the symbolic model;

False when a vulnerability is discovered, and the tool generates a concrete attack trace as evidence;

Timeout if the solver was not able to reach the goal within a set timeout;

“Cannot be proven” when the internal over-approximation leads to an attack derivation that cannot be translated into an executable trace.

In cases of inconclusive results, ProVerif still provides intermediate data that helps users manually pinpoint potential weaknesses and refine the model’s specification. By making non-essential adjustments, such as re-ordering process operations or introducing auxiliary checks, it may be possible to guide the tool’s search algorithm more effectively while maintaining the original semantics of the protocol. This refinement ensures that we do not settle for uncertainty, but rather channel the analysis toward a definitive proof of security or the identification of a legitimate attack.

Over the past two decades, extensive case studies have consistently affirmed ProVerif’s precision and efficiency in real-world applications. Its reliability is well-documented in the literature, notably in works such as Blanchet et al. (2016), Miculan et al. (2021), Miculan et al. (2023), Kobeissi et al. (2017), and Bhargavan et al. (2017), while the repository at Blanchet (2025) provides hundreds of additional practical examples demonstrating its broad utility across diverse security domains.

3.3 Example: Diffie-Hellman key exchange

In this section, we provide an example to illustrate the operational mechanics of ProVerif. We begin by formalising the classic Diffie-Hellman (DH) key exchange protocol, which is designed to allow two parties to establish a shared secret over an insecure channel. By encoding the protocol steps and the capabilities of a Dolev-Yao adversary into the applied π -calculus, we demonstrate that the unauthenticated version of the protocol is susceptible to a man-in-the-middle attack. In this scenario, ProVerif generates a counter-example (an attack trace) showing how an adversary can intercept the public values, substitute them with its own, and establish independent encrypted sessions with both legitimate parties.

To rectify this vulnerability, we extend the model by introducing digital signatures, transforming the exchange into an Authenticated Diffie-Hellman protocol. By requiring each party to sign their respective public contributions, a link is established between the ephemeral keys and the participants' long-term identities. We then re-verify the strengthened model using ProVerif. This second analysis proves that the secrecy of the shared key (and thus the authenticity of the session) is maintained even in the presence of an active attacker, thereby validating the effectiveness of the cryptographic solution and the utility of automated formal verification in detecting and resolving protocol flaws.

The Diffie-Hellman key exchange protocol, introduced by Diffie and Hellman (1976), represents the first practical method for two parties to establish a shared secret over an insecure communication channel without prior exchange of keys. The security of the protocol relies on the Discrete Logarithm Problem (DLP), which states that while exponentiation in a finite cyclic group is computationally efficient, inverting the operation is computationally infeasible for sufficiently large parameters.

The protocol operates within a cyclic group G with a generator g . The process begins with both parties, Alice and Bob, agreeing on these public parameters. Alice selects a private random integer a (her secret key) and computes her public value $A = g^a \pmod{p}$. Simultaneously, Bob selects a private random integer b and computes $B = g^b \pmod{p}$.

After exchanging these public values over the network, Alice computes the shared secret as $K_A = B^a \pmod p$, while Bob computes $K_B = A^b \pmod p$. Due to the properties of modular exponentiation, both calculations result in the same value:

$$K = K_A = (g^b)^a = g^{ba} = g^{ab} = (g^a)^b = K_B \pmod p$$

While a passive observer can see g , p , A , and B , she cannot derive K without solving the DLP to find a or b . However the protocol's lack of built-in authentication makes it vulnerable to an active attacker who can intercept and replace these public values in transit.

The implementation of the basic DH exchange in ProVerif is available in Listing 3.1. The script uses the applied π -calculus to model the algebraic properties of the protocol and test the confidentiality of the resulting shared secret.

The ProVerif model is structured into three main parts: the algebraic declarations, the security goals, and the process logic.

Since ProVerif does not natively understand modular exponentiation, we define a type element and exponent along with a function *exp*. The equation construct tells the solver that $\text{exp}(\text{exp}(g, x), y)$ is equivalent to $\text{exp}(\text{exp}(g, y), x)$: the commutativity allows both Alice and Bob to derive the same shared key.

Then, we also define a symmetric encryption scheme (*senc*, *sdec*) where the shared element serves as the encryption key. The equation for symmetric decryption ensures that a message encrypted with key k can be retrieved only with that same key.

The query `attacker(secretData)` command instructs ProVerif to search for any execution trace where the adversary (who controls the public channel c) can obtain the private bitstring `secretData`.

Then the process logic is defined:

1. Alice generates a fresh secret exponent x , broadcasts her public half gx , receives Bob's gy , and calculates $k = \text{exp}(gy, x)$;
2. Bob performs the mirrored operation;
3. then Alice attempts to send the secret data encrypted under k .

```

1  (* Types and constants *)
2  type element.
3  type exponent.
4  const g: element.
5
6  (* Diffie-Hellman math *)
7  fun exp(element, exponent): element.
8  equation forall x: exponent, y: exponent; exp(exp(g, x), y) =
    exp(exp(g, y), x).
9
10 (* Symmetric encryption definition *)
11 fun senc(bitstring, element): bitstring.
12 fun sdec(bitstring, element): bitstring.
13 equation forall m: bitstring, k: element; sdec(senc(m, k), k)
    = m.
14
15 (* Secrecy goal *)
16 free secretData: bitstring [private].
17 query attacker(secretData).
18
19 free c: channel.
20
21 let processAlice =
22 new x: exponent;
23 let gx = exp(g, x) in
24 out(c, gx);
25 in(c, gy: element);
26 let k = exp(gy, x) in
27 out(c, senc(secretData, k)).
28
29 let processBob =
30 new y: exponent;
31 let gy = exp(g, y) in
32 in(c, gx: element);
33 out(c, gy);
34 let k = exp(gx, y) in
35 0.
36
37 process
38 (!processAlice | !processBob)

```

Listing 3.1: ProVerif code for the unauthenticated Diffie-Hellman protocol.

To simulate a dynamic network, the main process block utilises the replication operator (!) to spawn an unbounded number of Alice and Bob sessions. This creates a parallel environment where the intruder can intercept, modify, or inject messages across different sessions.

By checking this code (`$ proverif dh.pv`), ProVerif identifies that while the underlying math is correct, the protocol lacks identity binding: since Alice has no way to verify that `gy` actually came from Bob, a Dolev-Yao attacker can perform an attack by intercepting the exchange and substituting their own public values, thus exchanging the key and decrypting the secret message (Listing 3.2).

By introducing digital signatures into the DH exchange, the protocol moves from anonymous to authenticated key agreement. The code available in Listing 3.3, when verified by ProVerif, can be used to prove that binding cryptographic identities to the ephemeral public values effectively fixes the previously identified attack.

The script extends the previous model with a formalisation of public key infrastructure and signature verification: the model defines a `sign` function and a corresponding `checksign` destructor. The equation `checksign(sign(m, k), pk(k), m) = m` ensures that a signature is valid only if it was generated by the secret key `k` corresponding to the public key `pk(k)`.

Moreover, in `processAlice`, Alice no longer sends just `gx`. Instead, she sends a tuple containing her public share and a digital signature of that share: `(gx, sign(gx, skA))`. This proves to any recipient that `gx` was intentionally sent by the owner of `skA`.

Before computing the shared secret, both parties now perform a conditional check using `if checksign(...)`. If an attacker attempts to substitute their own `gz`, the signature check will fail because the attacker cannot produce a valid signature using the legitimate participants' secret keys (`skA` or `skB`).

In the main process block, the public keys `pkA` and `pkB` are generated and broadcast on the public channel. This models a scenario where public keys are known to all parties (including the attacker), but the corresponding secret keys remain private.

```

1  [...]
2  -- Query not attacker(secretData[]) in process 1.
3  Translating the process into Horn clauses...
4  Completing...
5  Starting query not attacker(secretData[])
6  goal reachable: attacker(secretData[])
7
8  Derivation:
9  Abbreviations:
10 x_1 = x[!1 = @sid]
11
12 1. The message exp(g,x_1) may be sent to the attacker at output
13    {4}.
14    attacker(exp(g,x_1)).
15
16 2. Using the function g the attacker may obtain g.
17    attacker(g).
18
19 3. The message g that the attacker may have by 2 may be received
20    at input {5}.
21    So the message senc(secretData[],exp(g,x_1)) may be sent to the
22    attacker at output {7}.
23    attacker(senc(secretData[],exp(g,x_1))).
24
25 4. By 3, the attacker may know senc(secretData[],exp(g,x_1)).
26    By 1, the attacker may know exp(g,x_1).
27    Using the function sdec the attacker may obtain secretData[].
28    attacker(secretData[]).
29
30 5. By 4, attacker(secretData[]).
31    The goal is reached, represented in the following fact:
32    attacker(secretData[]).
33
34 [...]
35
36 The attacker has the message sdec(~M_1,~M) = secretData.
37 A trace has been found.
38 RESULT not attacker(secretData[]) is false.
39
40 -----
41 Verification summary:
42
43 Query not attacker(secretData[]) is false.
44
45 -----

```

Listing 3.2: ProVerif output for the unauthenticated Diffie-Hellman protocol, showing the trace of the found attack.

```

1  (* Types and constants *)
2  type element.
3  type exponent.
4  type skey.
5  type pkey.
6  const g: element.
7  fun e2b(element): bitstring [typeConverter].
8
9  (* Diffie-Hellman math + Symmetric encryption definition: [...] *)
10 (* Signature logic *)
11 fun pk(skey): pkey.
12 fun sign(bitstring, skey): bitstring.
13 fun checksign(bitstring, pkey, bitstring): bitstring.
14 equation forall m: bitstring, k: skey; checksign(sign(m, k), pk(k)
    , m) = m.
15 (* Secrecy goal *)
16 free secretData: bitstring [private].
17 query attacker(secretData).
18
19 free c: channel.
20
21 let processAlice(skA: skey, pkB: pkey) =
22   new x: exponent;
23   let gx = exp(g, x) in
24   out(c, (gx, sign(e2b(gx), skA)));
25   in(c, (gy: element, sig: bitstring));
26   if checksign(sig, pkB, e2b(gy)) = e2b(gy) then
27     let k = exp(gy, x) in
28     out(c, senc(secretData, k)).
29
30 let processBob(skB: skey, pkA: pkey) =
31   in(c, (gx: element, sig: bitstring));
32   if checksign(sig, pkA, e2b(gx)) = e2b(gx) then
33     new y: exponent;
34     let gy = exp(g, y) in
35     out(c, (gy, sign(e2b(gy), skB)));
36     let k = exp(gx, y) in
37     0.
38
39 process
40   new skA: skey; new skB: skey;
41   let pkA = pk(skA) in let pkB = pk(skB) in
42   out(c, pkA); out(c, pkB);
43   ((!processAlice(skA, pkB)) | (!processBob(skB, pkA)))

```

Listing 3.3: ProVerif code for the authenticated Diffie-Hellman protocol. The “Diffie-Hellman math” and “Symmetric encryption definition” sections are identical to the ones in Listing 3.1.

```

1      [...]
2
3      -- Query not attacker(secretData[]) in process 1.
4      Translating the process into Horn clauses...
5      Completing...
6      Starting query not attacker(secretData[])
7      RESULT not attacker(secretData[]) is true.
8
9      -----
10     Verification summary:
11
12     Query not attacker(secretData[]) is true.
13
14     -----

```

Listing 3.4: ProVerif output for the authenticated Diffie-Hellman protocol.

The ProVerif analysis of this model yields a true result for the query `attacker(secretData)`, meaning the secrecy is preserved (Listing 3.4).

Because the adversary cannot forge signatures, they are effectively forced into a passive role. While they can still intercept the messages, they can no longer inject themselves into the middle of the exchange.

Chapter 4

Interface-based threat models of coupled systems

Often, the security of modern distributed systems is not purely a matter of cryptographic strength, but increasingly a question of how heterogeneous components interact through their defined interfaces. As systems become more and more relying on a complex web of mutual dependencies, the traditional perimeter-based threat model becomes insufficient. This chapter introduces a formal interface-based methodology designed to capture the details of these interactions, focusing on the boundaries where distinct actors meet. To test this methodology in a real-world context, we apply it to the Italian Electronic Identity Card (*Carta d'Identità Elettronica*, or CIE) ecosystem. We begin by decomposing the CIE's architectural components and its underlying protocols, providing a formal representation of its communication flows.

Building upon this foundation, we then define a fine-granularity threat model that, unlike other adversary models, allows for a precise description of attacker capabilities at the interface level, accounting for both external intruders and compromised internal components. However, such modelling often leads to a combinatorial explosion of possible threat scenarios: we must thus employ some strategies for restricting the search space, ensuring that formal verification remains computationally feasible.

Finally, we formalise phishing attacks in our methodology: by treating the human-to-interface interaction as a formal protocol step, we show how our framework can detect social engineering vulnerabilities.

This chapter incorporates and builds upon research material and figures previously published in Van Eeden, Paier, and Miculan (2024), Paier, Van Eeden, and Miculan (2024), and Paier, Van Eeden, and Miculan (2026).

4.1 Interface-based modelling

In this analysis, we shift the conceptualisation of connected entities from a monolithic “black box” toward a modular, interface-centric abstraction. Building upon the formal methodology established by Jacomme et al. (2021), we model each device, from resource-constrained sensors to high-performance clusters, as a discrete collection of functional interfaces.

This abstraction is fundamental to accurately formalising the attack surface and characterising the mechanics of data flow within complex and heterogeneous ecosystems. By decomposing an entity into its constituent interfaces, we can define the precise boundaries of data exchange. Each interface serves as a gateway facilitating two distinct vectors:

Ingress i.e. the set of logical or physical entry points through which the entity consumes external stimuli, protocol messages, or commands;

Egress: i.e. the set of exit points through which the entity propagates state information, responses, or telemetry.

The primary advantage of this model is the recognition of independent compromise. In a multifaceted threat landscape, the security posture of an entity is rarely uniform across its entire architecture, since each interface may operate under distinct protocol stacks, cryptographic standards, and access control policies. An adversary may thus successfully breach a single interface while preserving the security of the others. This creates a state of partial system compromise, essential to consider when modelling lateral movement and privilege escalation.

More formally, we represent the system as a directed graph $G = (V, E)$. In our model, the set of nodes V represents the architectural components

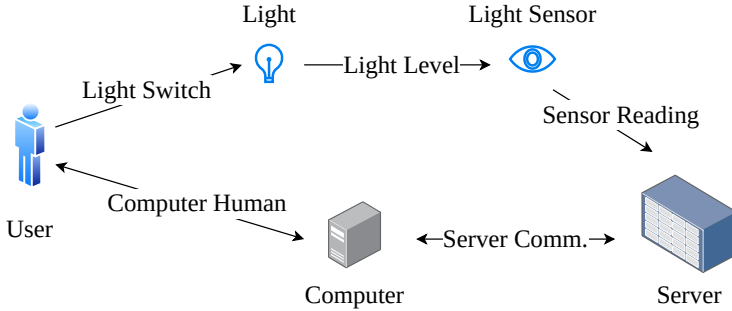


Figure 4.1: An example of a system interface model we can describe using our methodology.

of the system, such as hosts, identity providers, or human participants. Each node $v \in V$ is characterised by a set of interfaces that define its interaction boundaries.

The set of directed edges E represents the communication channels established between these interfaces. An edge $e = (u, v)$ signifies a data flow originating from the output interface of component u and terminating at the input interface of component v . This mapping is important for security analysis, as it establishes the information flow direction and defines the potential paths for adversary data exfiltration or tampering.

A schematic representation of a generic interface model, developed using our proposed methodology, is illustrated in Figure 4.1. The example model illustrates a system of components interacting through a series of specialised interfaces: the *User* interacts with the system via a bidirectional *Computer Human* interface to command a *Computer*, while also interacting physically with a *Light Switch* to actuate a *Light*. This actuation creates a *Light Level* signal that is captured by a *Light Sensor*, which then transmits it to a *Server* through a *Sensor Reading* interface. Moreover, the *Server* also has a bidirectional *Server Comm.* interface with the *Computer*, for example to show the light level to the human.

Notice the diverse kind of interfaces captured in the model, which encompasses both physical interactions (light propagation) and standard network protocols (e.g. the computer to server protocol).

The modelling process should begin with a coarse-grained view, where an entity might initially be defined with no exposed surfaces or with limited generic interfaces. As the design phase progresses, this monolithic entity is decomposed: existing interfaces may be split to distinguish between varying data flows (e.g. separating control signals from payload data), or new interfaces may be introduced to represent specific physical or logical endpoints previously abstracted away. This stepwise refinement ensures that the model remains manageable while progressively increasing its fidelity.

Refinement is also driven by the evolving threat landscape. An interface that was initially omitted may need to be explicitly modelled in two cases: if analysing the actual system implementation it is revealed that a theoretical boundary is, in practice, a distinct ingress/egress point (e.g. a debug port left open in production); or if new research uncovers attack methods that exploit specific interactions previously considered safe (e.g. fault injection on a power supply line).

Conversely, this refinement principle allows for the pruning of the model to maintain analytical efficiency. If an interface is verified to be secure and invulnerable to the relevant class of attacks, it can be omitted from the threat scenarios. This selective modelling focuses the analysis strictly on the “attackable” surface, preventing state-space explosion during formal verification while ensuring that vulnerabilities are not omitted.

4.2 Running example: CIE

To illustrate our methodology, we employ the Italian Electronic Identity Card (CIE) as a primary running example. As an eIDAS-compliant framework, the CIE encapsulates the architectural complexities typical of modern electronic identity (eID) ecosystems.

Consistent with other eIDAS-compliant implementations, the CIE authentication mechanism utilises the SAML 2.0 (Security Assertion Markup Language) standard. This facilitates the secure exchange of authentication and authorisation metadata across disparate security domains, primarily involving two core entities: firstly, the *Identity Provider* (IdP), which is

managed by the Italian Ministry of the Interior, and serves as the authoritative entity responsible for verifying user credentials and generating signed security assertions; secondly, the *Service Providers* (SPs), which are the federated public administration portals that rely on IdP assertions to grant access to protected resources.

The CIE framework establishes three authentication levels to access online services, categorised by the strength of the credentials required:

Level 1 implements single-factor authentication, relying exclusively on a traditional username and password credential pair.

Level 2 introduces multi-factor authentication (MFA) by requiring a registered secondary device. Users must either provide a one-time pass-code (OTP) transmitted via SMS or authorise the session through the *CieID* mobile application. The application-based flow involves out-of-band authorisation via push notifications or QR code scanning, followed by the entry of an app-specific PIN.

Level 3 represents the highest assurance level, requiring possession of the physical CIE smart card combined with its associated PIN. The smart card's secure element is accessed via Near-Field Communication (NFC) when interfaced with an NFC-enabled smartphone running the *CieID* application. Alternatively, when utilising a dedicated peripheral connected to a traditional PC, the system establishes a trusted channel via a Secure Messaging (SM) protocol.

To maintain a higher security level, service providers typically mandate Level 2 or Level 3 authentication for sensitive transactions. Level 1 is generally restricted to low-assurance scenarios involving minimal processing of personal data.

The official technical documentation for the CIE system (AGID 2026) presents a high-level logical schematic, reproduced in Figure 4.2. Translating this schematic directly into our interface-based formalism yields the preliminary abstraction shown in Figure 4.3. In this initial state, the model remains coarse-grained: it captures the essential actors, but the interactions are ambiguous and the topology is not yet strictly defined.

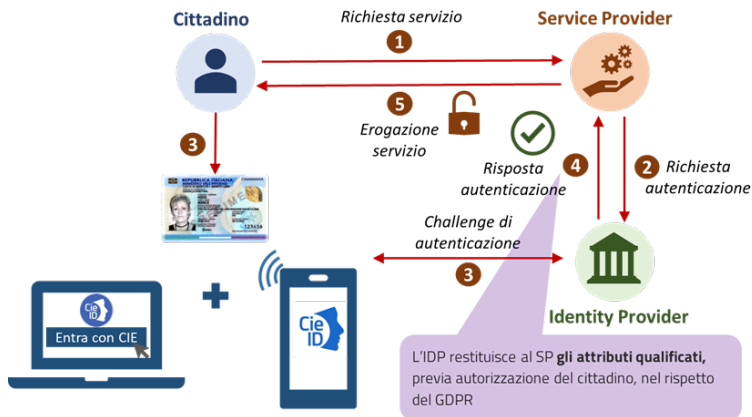


Figure 4.2: CIE interface model as described in the official documentation.

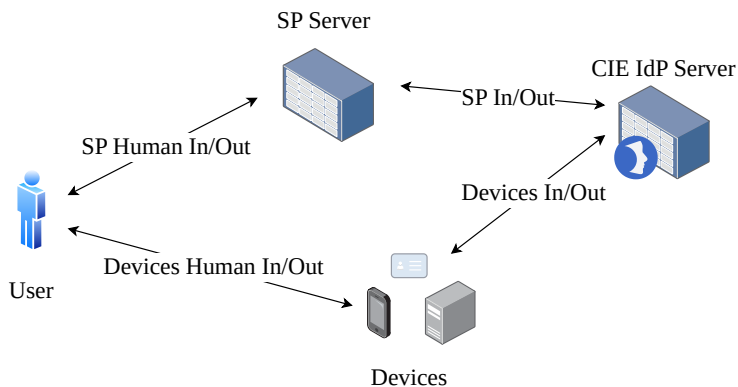


Figure 4.3: Initial CIE system interface model.

To achieve a higher degree of fidelity, we apply the refinement process. We first partition the interfaces according to their hosting hardware, clearly delineating the boundaries between the Computer, Smartphone, and Smart Card. Moreover, we refine the model's scope by excluding the Service Provider entity, since, while the SP is responsible for triggering the authentication request and consuming the final assertion, it acts as a passive observer during the core authentication process. This refined organisation is visible in Figure 4.4.

We then adjust the connectivity model to reflect reality: specifically, we recognise that the Smart Card does not communicate directly with the Identity Provider or the User. Instead, it functions as a peripheral that bridges exclusively through the Mobile device or the Computer. This corrected interaction model is illustrated in Figure 4.5.

Furthermore, a detailed analysis of the official documentation reveals a suite of different protocols handling the communication between the IdP and the Mobile device. To enhance the granularity of our model, these are decomposed into individual logical interfaces, such as those dedicated to Push notifications and SMS-based OTP delivery.

Simultaneously, the interface model for the computer is refined to explicitly differentiate user-facing peripherals into distinct ingress and egress channels (i.e. the physical keyboard and the display). This distinction is necessary from a security perspective, as an adversary may compromise these interfaces independently; for instance, a hardware or software-based keylogger grants an attacker access to the input stream without necessarily compromising the integrity of the visual output on the screen. These refinements are consolidated in Figure 4.6

Still, the current abstraction does not account for cognitive-layer exploits, such as phishing: to incorporate these we can apply an additional layer of refinement by dividing the user entity into two distinct components: the user's physical faculties (e.g. biological sensors and actuators like eyes and hands) and the user's cognitive state (the "mind").

These components are interconnected via a conceptual interface representing the user's decision-making process: phishing is modelled as a compromise of this interface, where an adversary manipulates the ingress

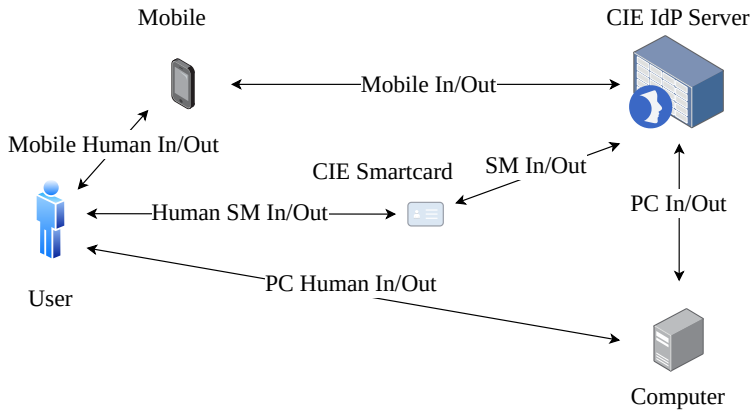


Figure 4.4: First refinement of the CIE system interface model. The devices are now separated.

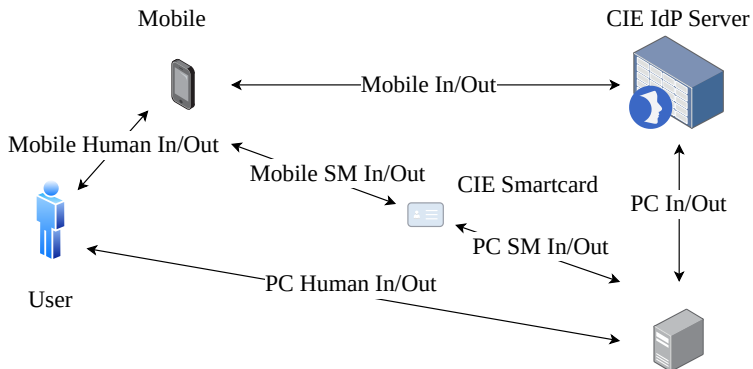


Figure 4.5: Second refinement of the CIE system interface model. The interactions with the Smart Card are now linked like in the implemented system.

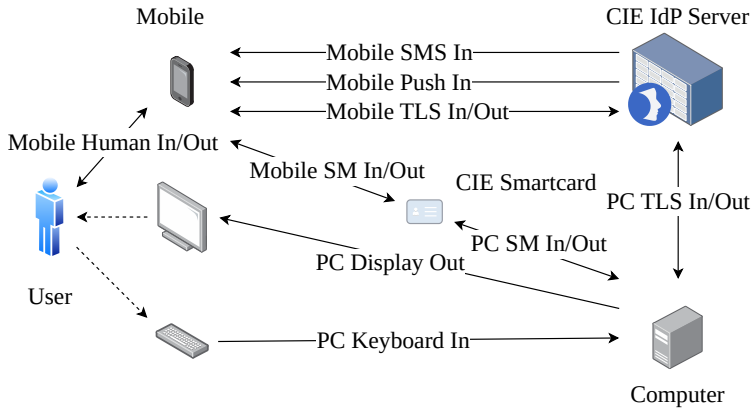


Figure 4.6: Final refinement of the CIE system interface model. The channels are separated into specialised ones.

data to deceive the user’s perception. In this scenario, the user performs an action that appears cognitively legitimate based on the perceived output (e.g. a fraudulent login screen), yet is functionally malicious. The final model can be found in Figure 4.7.

We may still simplify the model by re-evaluating the representation of the cognitive layer: in fact, the complexities of phishing can be captured without a dedicated “mind-to-sense” interface. This can be accomplished by instead defining a taxonomy of user profiles: for instance, distinguishing between an attentive user which is not easily deceived by malicious links, and a susceptible user which always fails to verify the legitimacy of a Service Provider’s URL.

The removal of the cognitive interface is further justified by current technical constraints: an external adversary cannot directly compromise the biological link between human perception and cognition through traditional cyber vectors. However, this abstraction remains flexible: should brain-computer interfaces (such as Neuralink) achieve widespread adoption, the link between the biological and the digital would become a viable attack surface. In such a future scenario, this interface would need to be reinstated to account for direct neural manipulation.

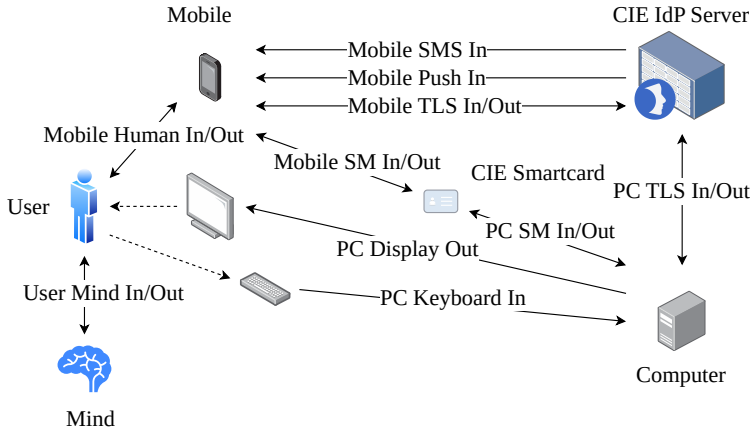


Figure 4.7: Further refinement of the CIE system interface model. The user cognitive layer is explicitly modelled.

Consequently, based on all these considerations, the final interface model for the CIE system, which serves as the basis for our formal analysis, is the configuration previously established in Figure 4.6.

The user's computer is modelled using the following interfaces:

PC Keyboard (pc-kbd) representing the physical input vector for user-driven data entry;

PC Display (pc-displ) representing the visual output vector; this interface is strictly limited to data egress;

PC TLS (pc-tls) denoting encrypted communication channels established with remote entities, such as the IdP server. This interface is modelled as a bidirectional primitive, as the compromise of a TLS session inherently affects both ingress and egress data streams;

PC SM (pc-sm) representing bidirectional communication with the CIE smart card via the *Secure Messaging* (SM) protocol. This interface accounts for the end-to-end encryption and authentication layers that operate independently of the physical transport medium (e.g. USB or Bluetooth), thus being a distinct logical interface.

Mobile devices are represented by the following set of interfaces:

Mobile Human (mob-hum) representing the unified interaction layer between the user and the device. Unlike the segregated keyboard and display interfaces of a PC, `mob-hum` characterises the touchscreen as a single bidirectional interface for both data ingress (input) and egress (output);

Mobile SMS (mob-sms) denoting the cellular radio module specifically utilised for the reception of Short Message Service (SMS) traffic from the IdP. This is modelled as a receive-only (ingress) channel;

Mobile Push (mob-psh) representing the logical interface for receiving push notifications via remote services (e.g. Firebase Cloud Messaging). This is also a receive-only channel;

Mobile TLS (mob-tls) defining encrypted communication channels established with remote entities, such as the IdP server. Consistent with PC TLS, this is treated as a bidirectional interface where the compromise of a direction implies the compromise of the session;

Mobile SM (mob-sm) representing bidirectional communication with the CIE smart card via the *Secure Messaging* (SM) protocol over Near-Field Communication (NFC). This interface models the logical security layer that persists regardless of the proximity-based physical transport.

The smart card is abstracted via two primary interfaces: one for bidirectional Secure Messaging with the computer and another for SM communication via NFC with the mobile device. The IdP server is modelled by its TLS interfaces, which facilitate secure communication with both PCs and mobile devices, supplemented by specialised egress interfaces for Mobile SMS and Mobile Push delivery.

A significant advantage of our interface-centric model is its inherent extensibility. It allows for the seamless integration of additional dedicated interfaces to represent emerging threat vectors. For instance, the model can be expanded to include specialised interfaces for side-channel analysis

or fault injection attacks, enabling the formalisation of information leakage or adversarial data injection within the system's boundary definitions.

4.2.1 Protocols

Building upon the architecture defined previously, the CIE framework specifies three protocols for Level 2 MFA and two for Level 3 MFA. The fundamental principles of these protocols are summarised below, with detailed sequence diagrams and technical specifications provided in Figures 4.8 to 4.12.

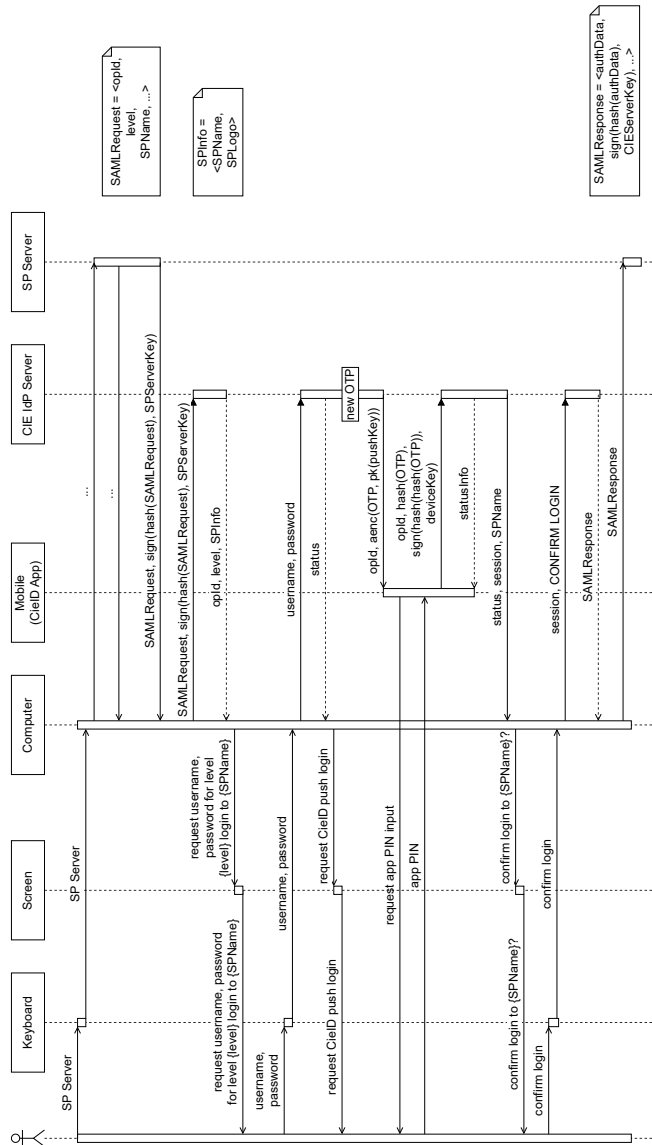
L2SMS (Figure 4.8) relies on the combination of standard login credentials and physical control over the SIM card linked to the user's account. Authentication is finalised by entering a One-Time Passcode received via SMS. Once the user initiates a connection request to a specific Service Provider, the Identity Provider prompts for their Level 1 credentials (username and password). Upon successful verification of these primary credentials, the IdP generates a unique OTP cryptographically bound to the current session and transmits it via SMS to the user's registered mobile device. The user then retrieves this OTP from the mobile device and manually enters it in the computer, which then transmits it back to the IdP for final validation. If the provided OTP matches the server-side record, the IdP issues the authentication assertion to the SP, thereby completing the two-factor authentication process.

L2PSH (Figure 4.9) requires knowledge of the CIE credentials and the 6-digit PIN of the CieID application, alongside physical possession of the registered mobile device. The user authorises the login through a push notification. When the user initiates the login process and selects the push notification method, the IdP server generates a unique session-bound access token. Following Level 1 credential verification, the server generates a 4-digit OTP, encrypts it using the public key associated with the user's registered mobile device, and dispatches it alongside the access token via a push notification service. The encrypted payload is decrypted on the mobile device after the user provides their 6-digit CieID app PIN. Once authorised, the CieID app computes a cryptographic hash of the OTP and

signs it. This signed hash, along with the access token and the digital signature, is transmitted back to the IdP. The server then validates the signature and confirms the OTP hash matches its internal record to grant access. The entire exchange is governed by strict temporal constraints: the session expires if a valid response is not received within a predefined window, mitigating the risk of replay or long-term interception.

L2QRC (Figure 4.10) authentication is performed by using the CieID app to scan a QR code displayed on the computer. This method demands possession of the registered smartphone and knowledge of the app's 6-digit device PIN. The authentication process begins when the user initiates a login request to the SP via their computer. In response, the IdP server generates a unique, session-specific challenge. This challenge is encoded into a QR code displayed on the computer's screen, containing metadata such as the operation ID and the SP's identity. To proceed, the user scans the QR code using the CieID application on their registered smartphone. After verifying the operation details, the user authorises the login by entering their 6-digit device PIN. The application then computes a cryptographic hash of the challenge and signs it using the device's private key. This digital signature, along with the user's CIE card number, is transmitted back to the IdP server. The server completes the process by validating the signature against the registered device certificate. If the signature is authentic and corresponds to the active session, access is granted. To mitigate session hijacking attacks, the challenge is protected by a strict two-minute expiration timer; if the flow is not completed within this window, the login attempt is automatically invalidated.

L3APP (Figure 4.11) uses an NFC-enabled smartphone as a bridge to interact with the physical CIE smart card. Security is rooted in the possession of the card and knowledge of the smart card's specific PIN. The authentication process begins with the IdP generating a session-specific QR code, containing the login metadata and the user's CIE card number, which is displayed on the computer's screen. The user scans this code using the CieID application, which then parses the request and presents the SP's identity for user verification. To authorise the transaction, the user must provide the final four digits of their CIE PIN (PIN2), while the



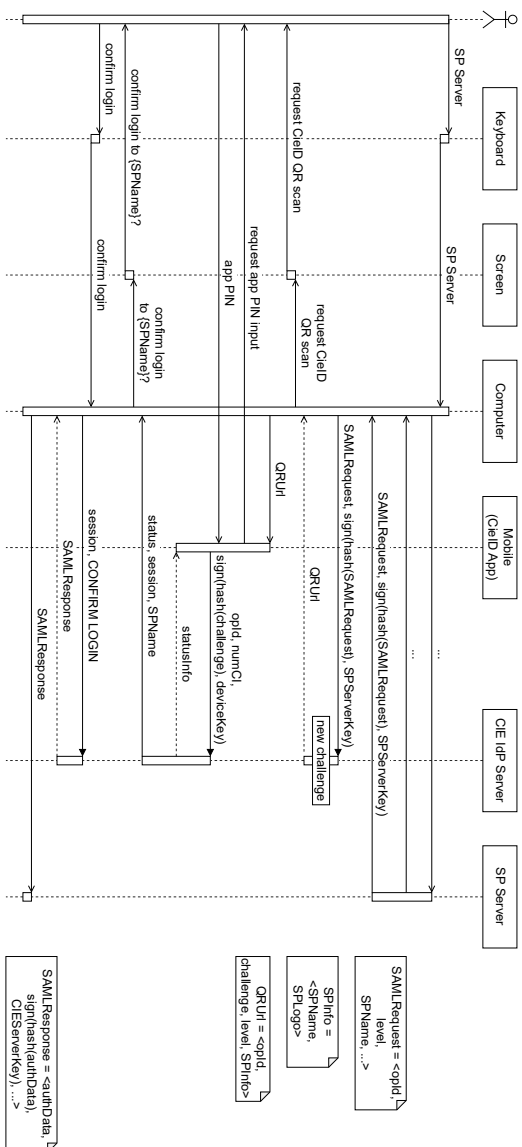


Figure 4.10: Sequence diagram of the L2QRC protocol, Level 2 authentication via QR code recognition.

first four digits (PIN1) are retrieved from the app's secure local storage. Once the full PIN is reconstructed and the user places their smartphone near the physical CIE smart card, the app establishes an NFC connection to unlock the card's cryptographic container. The app then reads the card's digital certificate and uses the card's internal private key to sign a hash of the QR code's URL. The app subsequently initiates a mutual TLS (mTLS) handshake with the IdP server, utilising the CIE card's certificate for client-side authentication. Within this secure tunnel, the app transmits the signed URL hash. Upon successful verification of the signature and the certificate's validity, the IdP generates an OTP and returns it to the app. Finally, the user manually enters this OTP into their computer; the computer relays it to the IdP for a final check, granting access upon successful validation.

L3DSK (Figure 4.12) utilises a PKCS#11-compliant reader directly connected to a computer, typically a USB smart-card reader or an integrated laptop card slot. It allows the computer to communicate with the CIE smart card once the card PIN is provided, effectively eliminating the mobile device from the authentication chain. Upon selecting this method, the IdP redirects the user's browser to a secure HTTPS endpoint that mandates a specific client certificate chain for access. This triggers the local CIE middleware, which prompts the user to enter the final four digits (PIN2) of their smart card PIN. The middleware combines this input with the first four digits (PIN1), stored in an encrypted format on the host computer, in order to unlock the cryptographic container of the CIE smart card. Once unlocked, the browser completes a mTLS handshake, using the smart card's private key to generate a digital signature, proving ownership of the CIE digital certificate to the server. An observation regarding this protocol is that during the TLS handshake, the CIE smart card certificate is transmitted in plaintext before the encrypted channel is fully established. Because this certificate contains the user's full name, fiscal code (revealing date and place of birth), and document ID, this behaviour presents a significant privacy risk. An eavesdropper on the local network or ISP level could intercept the sensitive personal data and correlate it with the specific login attempt.

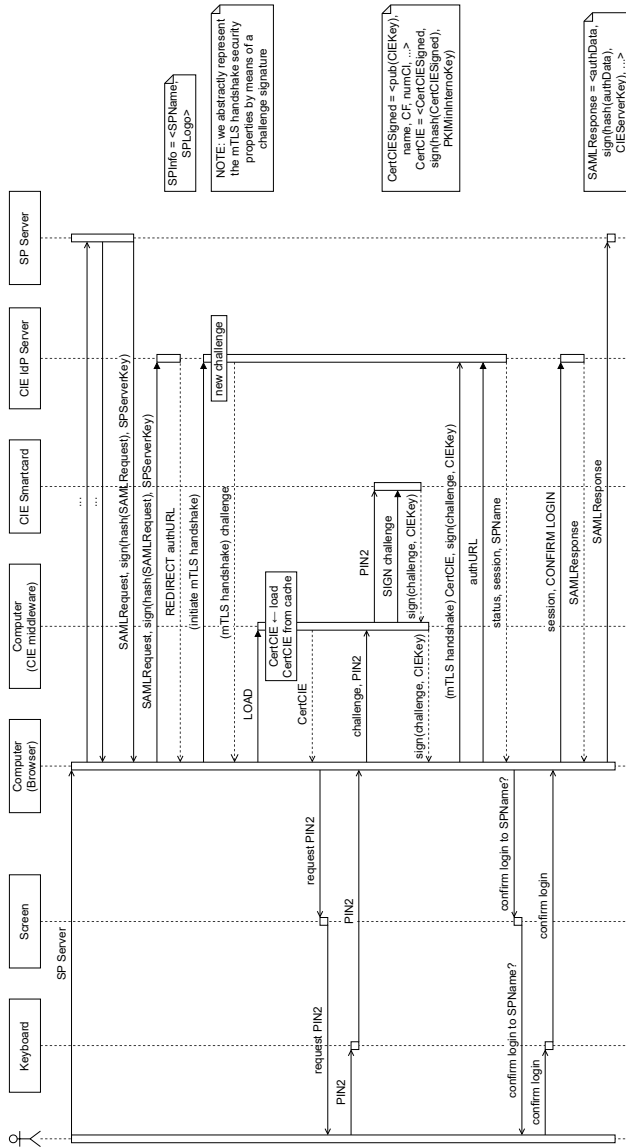


Figure 4.12: Sequence diagram of the L3DSK protocol, Level 3 authentication using standard TLS client certificates via the PKCS#11 CIE middleware.

4.3 Threat model

In this section, we delineate the methodology for modelling attacker interactions with the interfaces of a system, resulting in a diverse set of threat scenarios as formalised by Jacomme et al. (2021).

As already discussed, each interface within the architecture (for example, in the CIE case, see Figure 4.6) may support two distinct and independent information flows: input and output. When an interface is compromised, the attacker’s capabilities are defined by their level of access to these flows. Specifically, an adversary may:

Observe (Read) the data through an interface: passively monitor the data stream to extract information;

Alter (Write) data passing an interface: actively introduce or modify data, enabling the generation of arbitrary messages;

Fully Control (Read/Write) an interface: both intercept and manipulate the information flow.

Our interface-based methodology prioritises the analysis of the functional consequences of a breach, specifically addressing the question of *what happens if* an interface is compromised, rather than focusing on the technical particulars of the exploit (the *how*). This abstraction facilitates a consistent threat modelling framework that is agnostic to the specific vulnerabilities used by the adversary.

The model well represents varying degrees of adversarial control: a root-level malware infection on a primary actor (e.g. the PC or the Identity Provider) is modelled by granting the attacker unrestricted read-write access to all interfaces associated with that actor; a side-channel attack that captures plaintext communication between the PC and the smart card is represented by granting the attacker read-only access to the relevant output interface; a fault injection attack that permits the alteration of smart card output is modelled as write-only access to the corresponding device input interface.

Under this formalisation, an *uncompromised system* is defined as the one where the attacker possesses no access rights to any interface. Conversely,

a *fully compromised system* occurs when the attacker gains total read/write access across the entire interface set.

More formally, to provide a rigorous mathematical foundation for our security analysis, we formalise the attacker’s capabilities using the notation $\mathcal{A}_{d:a}^{if}$. This expression defines the degree of adversarial control exerted over a specific interface if , structured according to two parameters:

- $d \in \{\text{in}, \text{out}, \text{io}\}$ specifies the operational directionality under adversarial control: ingress (in), egress (out), or a bidirectional combination of both (io).
- $a \in \{\text{RO}, \text{WO}, \text{RW}\}$ characterises the specific access privileges obtained: read-only (RO) for observation, write-only (WO) for data alteration, or read-write (RW) for comprehensive manipulation.

Using this foundation, a *threat scenario* $\mathcal{S}$ is defined as a configuration wherein the attacker acquires distinct access control levels $d_k : a_k$ across a subset of system interfaces $\{if_1, \dots, if_n\}$.

This is formally represented by the set:

$$\mathcal{S} = \{\mathcal{A}_{d_1:a_1}^{if_1}, \dots, \mathcal{A}_{d_n:a_n}^{if_n}\}$$

This notation allows for precise mapping of different attack vectors. For example, the scenario $\{\mathcal{A}_{\text{out:RW}}^{\text{pc-displ}}, \mathcal{A}_{\text{in:RO}}^{\text{mob-hum}}\}$ describes a potential attack where the adversary possesses full control over the computer’s display output while simultaneously maintaining the ability to passively monitor the ingress of the mobile device’s touchscreen. Using our notation, the baseline security state, an entirely uncompromised system, is denoted by the empty set $\{\}$.

In principle, assigning varying levels of access, specifically none, read-only (RO), write-only (WO), or read-write (RW), to both directions of every interface generates a vast combinatorial space of potential threat scenarios. Within the CIE architecture illustrated in Figure 4.6, we identify 14 distinct information flows, considering that unidirectional interfaces, such as the keyboard and display, contribute only a single flow each. This configuration results in $4^{14} = 268\,435\,456$ unique adversarial combinations to be considered.

Protocol	pc-kbd	pc-displ	pc-tls	pc-sm	mob-hum	mob-sms	mob-psh	mob-tls	mob-sm	Theoretical number of scenarios	Reduced number of scenarios
L2SMS	✓	✓	✓		✓	✓				16 384	405
L2PSH	✓	✓	✓		✓		✓	✓		262 144	1215
L2QRC	✓	✓	✓		✓			✓		65 536	405
L3APP	✓	✓	✓		✓			✓	✓	1 048 576	1215
L3DSK	✓	✓	✓	✓						4096	81
Total										1 396 736	3321

Table 4.1: Number of theoretical and reduced scenarios for each CIE protocol.

This figure represents the theoretical upper limit of combinatorial test cases: in practice we achieve a significant reduction in complexity by focusing exclusively on the interfaces actively used by a given protocol.

For example, when analysing the L2SMS protocol, we can exclude attack vectors targeting the mobile TLS, mobile Push, and Secure Messaging interfaces of both the computer and mobile device, as these components are not used during this specific authentication flow. By isolating only the participating interfaces, the search space for the L2SMS protocol is reduced to $4^7 = 16\,384$ scenarios. This pruning is applied analogously to the remaining four protocols, ensuring the analysis remains both computationally feasible and focused on relevant attack surfaces.

Table 4.1 details the specific interfaces involved in each Level 2 and Level 3 protocol, alongside the theoretical number of potential attack scenarios; the final column is reserved for subsequent discussion.

This table provides a comparative overview of the attack surfaces presented by the five protocols. Notably, L3DSK exhibits the most constrained attack surface due to its minimal interface requirements, as it bypasses the need for a mobile device. In contrast, L2PSH and L3APP use a significantly wider range of interfaces, thus expanding the potential vectors for compromise. Generally, a broader attack surface increases the likelihood that an adversary can find and exploit a successful attack path.

4.4 Restricting the search space

The data in Table 4.1 reveals a cumulative total of approximately 1.4 million potential attack scenarios across all protocols. While this represents a significant reduction from the initial theoretical maximum, the complexity remains substantial. To refine this search space into a more manageable set of high-fidelity test cases, we can leverage two key observations.

First, acquiring write access to a system interface typically demands elevated privileges or the exploitation of specific platform-level vulnerabilities; such conditions would inherently grant the adversary read access as well. Consequently, write-only (WO) scenarios are considered inapplicable to the CIE model, as the ability to inject data invariably implies the capability to monitor the same interface. By excluding WO configurations, we can effectively constrain the analysis to three distinct access control levels: none, Read-Only (RO), and Read-Write (RW).

Secondly, the complexity of compromise varies significantly between data directions: gaining control over an interface's input (ingress) is generally less challenging than controlling its output (egress). This disparity is primarily due to the asymmetric permission models implemented in modern operating systems.

For example, reading data from a USB peripheral often requires only standard user-level execution. Conversely, intercepting, suppressing, or modifying the output data transmitted over that same USB bus by a separate, authorised process typically necessitates administrative privileges or the exploitation of kernel-level vulnerabilities. Consequently, we can apply a hierarchical assumption to our threat model: an adversary capable of achieving read-write (RW) control over an interface's output can be functionally assumed to possess equivalent control over its input. This directional dependency further excludes redundant scenarios where an attacker is modelled as powerful enough to manipulate outgoing data but somehow incapable of seeing what comes in.

Based on this hierarchy of complexity, we can establish a *strength ordering* for access control levels. In this model, gaining read-write control over an interface's output (out:RW) is the highest tier of compromise, as it

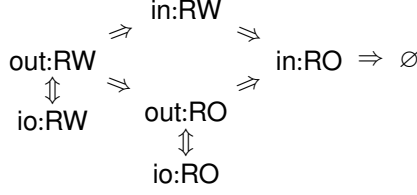


Figure 4.13: Security implication between access control levels.

is considered strictly more difficult to achieve than either read-only access to that output (out:RO) or read-write access to its input (in:RW).

Both out:RO and in:RW are, in turn, considered stronger than simple read-only access to the input (in:RO), which represents the most basic level of compromise. The least element of this ordering is the absence of any control access.

More formally, we define a partial order $\Rightarrow$ over the set of access control levels. In this framework, the expression $\ell_1 \Rightarrow \ell_2$ signifies that an adversary capable of achieving access level ℓ_1 inherently possesses the capabilities granted by access level ℓ_2 . Furthermore, two access levels ℓ_1 and ℓ_2 are considered equivalent, denoted as $\ell_1 \Leftrightarrow \ell_2$, if and only if they mutually imply each other (i.e. if $\ell_1 \Rightarrow \ell_2$ and $\ell_2 \Rightarrow \ell_1$). The order is shown in Figure 4.13, where $\emptyset$ represents no access at all.

This order can be pointwise extended to scenarios:

$$\mathcal{S}_1 \Rightarrow \mathcal{S}_2 \stackrel{\Delta}{\iff} \forall \mathcal{A}_{\ell_2}^{if} \in \mathcal{S}_2. \exists \mathcal{A}_{\ell_1}^{if} \in \mathcal{S}_1. \ell_1 \Rightarrow \ell_2$$

Consequently, these implications allow for significant optimisation in the formal verification process:

- if a protocol is proven secure under a specific attack scenario $\mathcal{S}$, it is necessarily secure under any weaker or equal scenario $\mathcal{S}'$ ($\mathcal{S} \Rightarrow \mathcal{S}'$). In this case, we state that *security in $\mathcal{S}'$ is implied by a stronger scenario*.
- if a vulnerability is identified in scenario $\mathcal{S}$, that vulnerability persists in any scenario $\mathcal{S}'$ that is stronger or equal ($\mathcal{S}' \Rightarrow \mathcal{S}$). In this case, we state that *the vulnerability in $\mathcal{S}'$ is implied by a weaker scenario*.

By applying these logical bounds, we can reduce the search space in both directions: once an attack is discovered in a specific scenario, all stronger scenarios are discarded as trivially vulnerable. Likewise, if a protocol is verified as secure in a given scenario, all weaker configurations are discarded as trivially secure.

By leveraging these implications, we can observe that for each single direction interface (such as keyboard and display) there are three possible access control levels: none, RO, RW. On the other hand, bidirectional interfaces admit five possible access control levels. Overall, this leads to an upper bound of $3 \cdot 3 \cdot 3 \cdot 3 \cdot 5 \cdot 5 \cdot 5 \cdot 5 \cdot 5 = 253\,125$ scenarios for a protocol that would use all interfaces.

By leveraging these implications, we can refine the possible access control levels for each interface type. For unidirectional interfaces (such as the keyboard and display), there are three levels: none, RO, RW. For bidirectional interfaces, the strength ordering and the exclusion of write-only access reduce the state space to five meaningful configurations (e.g. none, in:RO, in:RW, out:RO, and out:RW).

Consequently, this logic establishes a new theoretical upper bound of $3 \cdot 3 \cdot 3 \cdot 3 \cdot 5 \cdot 5 \cdot 5 \cdot 5 \cdot 5 = 253\,125$ scenarios for a hypothetical protocol utilising every available interface. This represents a significant optimisation from the original combinatorial explosion.

We can further reduce the search space by recognising that for TLS channels, the only relevant access levels are none, io:RO and io:RW. This is because the cryptographic properties of a TLS tunnel ensure that gaining access to one direction of the communication channel inherently implies equivalent access to the other. In our analysis, we utilise a simplified TLS model that captures the core security properties of TLS (i.e. confidentiality, authentication, and message order) while acknowledging that an adversary may still block or delay packets.

A similar principle applies to the SM channels established between the computer (or mobile device) and the smart card. Since these channels are logically bidirectional and protected as a single session, we treat them as unified io interfaces. By applying this constraint, the theoretical upper bound for a protocol using all interfaces drops to $3 \cdot 3 \cdot 3 \cdot 3 \cdot 3 \cdot 3 \cdot 3 \cdot 5 = 32\,805$.

Finally, we consider that each specific protocol utilises only a subset of the available system interfaces. By restricting our analysis to the interfaces active within a given protocol’s flow, we achieve a final reduction in complexity. For the L2SMS protocol, for instance, the number of relevant scenarios is reduced to $3 \cdot 3 \cdot 3 \cdot 3 \cdot 5 = 405$. Applying this refined logic across the entire suite of CIE protocols yields the finalised figures presented in the last column of Table 4.1, resulting in a cumulative total of 3 321 scenarios.

4.5 Phishing attacks

Despite their inherent security, MFA systems remain susceptible to human error and social engineering. It is therefore important to evaluate the resilience of the CIE’s MFA mechanisms against such threats.

Phishing attacks, using vectors such as SMS or spoofed websites, pose a significant risk to the authentication process. In these scenarios, users may be deceived into disclosing sensitive information, such as OTPs. To formalise this within our model, we adopt the assumption that the adversary controls the targeted server, i.e. the Service Provider the user intends to authenticate with. This approach is consistent with the methodology established in Jacomme et al. (2021); however, as the CIE login process does not utilise biometric data for remote authentication, fingerprint-related bypasses are excluded from our analysis.

The CieID app attempts to mitigate these risks by displaying a three-second warning message that prompts users to verify the displayed URL against the official CIE IdP address. Nevertheless, users may overlook or disregard this warning, for instance due to urgency. Furthermore, sophisticated techniques such as IDN homograph attacks can exploit internationalised domain names to create visually indistinguishable URLs, effectively deceiving even vigilant users (Holgers et al. 2006).

To formally represent a phishing attack scenario we introduce the symbol PH into our attack notation. In our model, a scenario marked with PH implies the adversary has compromised the cognitive layer of the interaction via a malicious SP URL. This inclusion doubles the number of scenarios to be analysed, resulting in a total of 6 642 unique test cases.

Chapter 5

Fine-grained formal analysis of protocols

In this chapter we continue building upon the work previously established in Van Eeden, Paier, and Miculan (2024), Paier, Van Eeden, and Miculan (2024), and Paier, Van Eeden, and Miculan (2026), detailing the formal modelling of protocols using ProVerif, a state-of-the-art automated security protocol verifier based on the Dolev-Yao model.

Our analysis focuses on the CIE Level 2 and Level 3 MFA protocols as our running example, evaluating their resilience against diverse adversarial conditions, involving pre-compromised user passwords, varying degrees of attacker control over system interfaces, and successful phishing attempts. By encoding these threats within our formal model, we provide a comprehensive assessment of the CieID multi-factor architecture’s effectiveness in protecting user accounts.

We examine whether the security objectives hold even when individual components suffer interface compromise. The complete formalisation, including the ProVerif scripts and model definitions, is publicly available at Paier et al. (2025).

5.1 Modelling CIE protocols in attack scenarios

In ProVerif, protocol participants are modelled as parallel-executing processes that communicate by exchanging messages over *channels*. This concurrent execution allows the tool to explore all possible interleavings of actions an attacker might exploit.

The fundamental primitives for communication are:

- $\text{out}(c, m)$: Represents the act of sending message m over channel c ;
- $\text{in}(c, x); P$: Represents a process that waits to receive a message from channel c , binds that value to the variable x , and subsequently proceeds with the execution of process P .

Channels are declared as either *public* or *private*. Adhering to the Dolev-Yao adversary model, the attacker possesses total control over the network. They can intercept, modify, or inject any message transmitted over channels known to them. While a private channel is a priori secure, it becomes accessible to the attacker if its name is disclosed or leaked during protocol execution.

Each interface within the architecture shown in Figure 4.6 is represented as a pair of private channels: one dedicated to input and the other to output. Additionally, we define a standard public channel *public*, which is always accessible to the adversary. For a given protocol formalisation P and an attack scenario $\mathcal{S}$, we derive a formalisation $P_{\mathcal{S}}$. This transformation incorporates new primitives that reflect the specific adversarial capabilities granted by $\mathcal{S}$:

- **Read-Only Access (RO)**: To simulate an attacker capable of eavesdropping on an interface without modifying its traffic, we mirror all messages transmitted on the corresponding private channels to the public channel. Formally, if *iface* is a private interface channel:

- Every instance of $\text{out}(\textit{iface}, \textit{message})$ is replaced by:

$$\text{out}(\textit{public}, \textit{message}); \text{out}(\textit{iface}, \textit{message});$$

- Every instance of $\text{in}(\text{iface}, \text{message})$ is followed by a disclosure:

$\text{in}(\text{iface}, \text{message}); \text{out}(\text{public}, \text{message});$

- **Read-Write Access (RW):** To grant the attacker complete control (including interception, suppression, and injection) we disclose the private channel name at the start of the process:

$\text{out}(\text{public}, \text{iface});$

Modelling authenticated communication channels, specifically those secured by TLS, requires specific abstraction techniques:

- We omit the low-level details of the TLS handshake (referencing Bhargavan et al. (2017) for dedicated analysis). Instead, we model a secure, authenticated channel using a private constructor function: $\text{TLS}(\text{id}, \text{id}) : \text{channel}$. Only authorised entities can invoke this function to derive unique, unguessable channel names. When a connection is established, the host generates a fresh session identifier, which is implicitly bound to all subsequent messages to prevent session confusion or cross-protocol data mixing.
- Because ProVerif’s native channels are synchronous, a direct implementation would prevent the simulation of active network attacks like message blocking or out-of-order delivery. To enable these behaviours, we decouple communications using the π -calculus parallel composition operator, effectively modelling the network as a set of independent message buffers.
- To accurately simulate scenarios where the adversary compromises a communicating party, we introduce an auxiliary TLS Manager process. This entity is responsible for establishing legitimate TLS channel names with the compromised identity and explicitly disclosing these names to the attacker, facilitating targeted session-interception attacks against honest participants.

Finally, to simulate phishing attacks, the model is configured to retrieve the target server’s URL from a public channel, mimicking an environment where a user may be directed to a malicious endpoint through compromised or manipulated online sources. In a legitimate authentication flow, a vigilant user is expected to compare the displayed URL against the official, trusted address of the IdP server.

However, to evaluate the system’s resilience, our model assumes that in scenarios marked with the PH attribute, the targeted user bypasses this verification step, allowing us to formally test whether the underlying protocol’s cryptographic bindings can still protect the user even when the initial human-centric verification fails.

5.2 Modelling security properties

The primary security property we evaluate is the *injective correspondence* between session initiation and successful authentication. This property ensures a strict one-to-one mapping: every login authorisation issued by the IdP to a specific device must correspond to a unique, *intentional* login request previously initiated by the user on that same device.

In ProVerif, this is formalised using an injective query that tracks the session identifier x :

query $x : id; \text{inj-event}(\text{Login}(x)) \Rightarrow \text{inj-event}(\text{UserLoginRequest}(x))$

In this specification:

- the $\text{UserLoginRequest}(x)$ event is triggered by the user process at the moment they intentionally start a login sequence;
- the $\text{Login}(x)$ event is raised by the IdP server only after it has successfully validated all required factors and credentials.

By using the `inj-event` keyword, ProVerif is able to verify that an attacker cannot perform a replay attack or a session injection, where a single user request could be manipulated to produce multiple successful logins, or where a login is granted without a corresponding user-initiated trigger.

In some instances, ProVerif may be unable to verify the injective correspondence goal, resulting in either a computational timeout or a “Cannot be proven” status. In such cases, we employ a strategy to simplify the proof search and guide the verifier toward discovering a concrete attack trace. By relaxing the property from injective to non-injective correspondence, we remove the requirement for a unique one-to-one mapping and focus solely on whether a login can occur without a preceding request. We modify the security query as follows:

$$\text{query } x : id; \text{event}(\text{Login}(x)) \Rightarrow \text{event}(\text{UserLoginRequest}(x)).$$

Where both $\text{Login}(x)$ and $\text{UserLoginRequest}(x)$ are defined as above.

This modified query compels ProVerif to search for an attack scenario where a successful login (Login event) occurs without any preceding initiation by the legitimate user (UserLoginRequest event). This approach is justified by the fact that finding a case where a Login happens without a prior UserLoginRequest is a more restrictive search for the attacker than the original injective goal, which only required finding a mismatch in the count of events: if ProVerif discovers an attack trace satisfying this simplified query, it automatically demonstrates a violation of the original injective correspondence goal, thus indicating a protocol vulnerability.

On the other hand, a positive verification outcome, i.e. where no attack is found for the non-injective query, does not necessarily guarantee the protocol’s security regarding the original goal. The injective property could still be violated in ways that are not captured by the more specific query, for instance by a replay attack, where an attacker uses a single legitimate UserLoginRequest to trigger multiple successful Login events.

5.3 Dynamic generation and exploration of attack scenarios

As detailed in Section 4.3, a comprehensive security evaluation of the CIE protocols requires testing them against an exhaustive array of attack scenarios. These scenarios represent the full range of control an adversary

```

results ← new map[Scenario → Result][{}]
for all sc ∈ Scenario do
  if results[sc] ≠ ⊥ then
    continue
  end if
  res ← VERIFY(sc)
  if res = TIMEOUT ∨ res = CANNOTBEPROVEN then
    res' ← VERIFYNOINJ(sc)
    if res' = VULN then
      res ← VULN
    end if
  end if
  results[sc] ← res
  if res = VULN then
    for all sc' ∈ Scenario s.t. sc' ⇒ sc do
      results[sc'] ← res
    end for
  else if res = SAFE then
    for all sc' ∈ Scenario s.t. sc ⇒ sc' do
      results[sc'] ← res
    end for
  end if
end for

```

Figure 5.1: Pseudocode for the dynamic exploration of attack scenarios.

might gain over system interfaces, ranging from passive eavesdropping to active interception, and include human-centric threats like phishing.

To manage this complexity, for each protocol formalisation P , we generate and analyse a specific variant P_S for every scenario S identified in our threat model. As discussed at the conclusion of Section 4.5, this approach results in the generation of over 6 600 distinct ProVerif models.

To ease the generation process of these numerous ProVerif models, we have developed a general *model template* for each protocol utilising the m4 macro processor (Kernighan et al. 1977). Within this template, each level of attacker control and human-induced threat is associated with a specific set of preprocessor definitions.

We utilise a Python script to systematically iterate through the entire range of possible scenarios; the corresponding pseudocode for this automation is presented in Figure 5.1. For each scenario, the script employs m4 to generate the corresponding ProVerif model instance, which is then automatically verified. To enhance efficiency, the script incorporates an optimisation that skips the verification of models whose outcome can be logically inferred from previously analysed scenarios:

- if P_S is found to be vulnerable, we can skip the analysis of $P_{S'}$ for all S' stronger than S ($S' \Rightarrow S$), as they are inherently vulnerable;
- if P_S is found to be secure, we can skip the analysis of $P_{S'}$ for all S' weaker than S ($S \Rightarrow S'$), as they are inherently secure.

5.4 Analysis results

This section presents the results of our analysis regarding the security properties of the CIE Level 2 and Level 3 authentication protocols. Additionally, we evaluate the scope and inherent limitations of formal proofs conducted within the symbolic model.

5.4.1 Level 2 protocols: L2SMS, L2PSH, L2QRC

While CIE Level 2 protocols offer a more robust defence against unauthorised access than simple password-based authentication, they still possess specific vulnerabilities. Table 5.1 details the findings of our formal analysis regarding these authentication schemes.

L2SMS While L2SMS authentication provides resistance against brute-force attacks targeting stolen credentials, it remains susceptible to more sophisticated adversarial techniques. For instance, malware or USB key-loggers pose a substantial threat by intercepting the one-time passcodes delivered via SMS: an attacker who successfully captures these codes can bypass the L2SMS mechanism entirely.

Threat Scenario					L2SMS	L2PSH	L2QRC
					✓	✗	✓
$\mathcal{A}_{in:RO}^{mob-sms}$					✗	—	—
$\mathcal{A}_{in:RW}^{mob-hum}$					✓	✗	✗
$\mathcal{A}_{io:RO}^{mob-hum}$					✗	✗	✓
$\mathcal{A}_{out:RO}^{pc-displ}$					✗	✗	✓
$\mathcal{A}_{out:RW}^{pc-displ}$					✗	✗	✗
$\mathcal{A}_{in:RO}^{pc-kbd}$					✗	✗	✓
$\mathcal{A}_{in:RW}^{pc-kbd}$					✗	✗	✗
$\mathcal{A}_{io:RO}^{pc-tls}$					✗	✗	✓
$\mathcal{A}_{io:RO}^{pc-tls}$					—	✗	✓
$\mathcal{A}_{io:RW}^{pc-tls}$					✗	✗	✗
PH					✗	✗	✗

Results:

- ✓ security proved
- ✓ security implied by the security proved in a stronger scenario
- ✗ attack found
- ✗ attack found in a weaker scenario
- — threat scenario not applicable to the protocol

Scenarios:

- PH: Phishing Attack
- $\mathcal{A}_{d:a}^{if}$: Attacker control over a system interface, where:
 - if is the name of the system interface
 - d represents control over interface's inputs (in), outputs (out) or both (io)
 - a is the access control over the interface: read-only (RO) or read-write (RW)

Table 5.1: Analysis results of Level 2 CIE protocols under different threat scenarios.

Furthermore, L2SMS possesses inherent transport-level security limitations when compared to protocols that leverage secure TLS channels. *SIM swapping* attacks, where an attacker transfers a victim’s phone number to a controlled SIM card, represent a critical vulnerability. Additionally, L2SMS may be compromised via SS7 attacks targeting legacy networks, which exploit architectural weaknesses in the signalling system and the weaker encryption protocols of these networks (Ullah et al. 2020).

Our analysis identifies a significant structural vulnerability in the L2SMS protocol: the manual entry of the one-time passcode delivered via the cellular network on the computer displays it within the user interface. This exposes the credential to e.g. malware capable of capturing video output, effectively compromising the second factor of authentication.

Currently, the most significant issue is that CIE users cannot disable L2SMS as an authentication method even after registering for more robust Level 2 protocols. This architectural choice maintains a persistent attack vector that reduces the security that would originate from the user’s adoption of stronger mechanisms. Finally, the fact that the enrolment process for new L2QRC devices itself relies on an SMS-based verification method raises concerns regarding the overall efficacy of the security upgrade path.

L2QRC The L2QRC protocol provides enhanced security compared to L2SMS, successfully resisting passive data extraction across nearly all examined interfaces. This robustness is demonstrated by the security of scenario $\{A_{lo:RO}^{pc-tls}, A_{in:RO}^{pc-kbd}, A_{out:RO}^{pc-displ}, A_{lo:RO}^{mob-hum}, A_{lo:RW}^{mob-tls}\}$, which remains secure even under full compromise of the Mobile TLS interface. However, the protocol proves vulnerable to active attackers who can manipulate system interfaces, e.g. to substitute (through direct or indirect control) the login operation identifiers presented to the CieID app.

These vulnerabilities originate from a lack of session-specific context: the CieID app does not present identifying elements, such as device fingerprints (Alaca et al. 2016), during the authentication phase. This prevents even security-conscious users from detecting concurrent or unauthorised login attempts and aborting the authentication session. Our analysis specifically finds an attack vector in scenario $\{A_{in:RW}^{pc-kbd}\}$. By intercepting

and modifying keyboard input, an attacker can silently alter the URL the user intends to visit, redirecting them to a malicious endpoint. While functionally similar to phishing, this method operates at a lower architectural level: rather than relying on social engineering to deceive the user, it directly manipulates the input stream, bypassing user awareness.

L2PSH The L2PSH protocol exhibits the weakest security properties among the evaluated methods. Our analysis demonstrates that an attacker can successfully complete a Level 2 login by possessing only Level 1 credentials, without requiring the compromise of any system interface.

Indeed, ProVerif identified a successful attack trace even within the empty scenario (representing the uncompromised system), indicating a fundamental logical flaw in the protocol’s design rather than a failure of a specific security channel. The high-level sequence diagram of this vulnerability is illustrated in Figure 5.2.

This vulnerability is similar to the one exploited by the timing attack identified in Google’s G2OT system, as detailed in Jacomme et al. (2021).

However, L2PSH may be significantly more susceptible due to its handling of concurrent authentication attempts: the CieID app automatically overwrites previous notifications with the most recent request. This behaviour effectively masks potentially fraudulent duplicate login attempts, hindering the user’s ability to recognise and reject a malicious concurrent session. Furthermore, the L2PSH request lacks context, such as device fingerprints: users have no means to verify the legitimacy of the request, substantially increasing the risk of successful account compromise.

To circumvent L2PSH authentication, an adversary only needs to monitor network traffic to detect a legitimate access attempt initiated by the user. Upon detection, the attacker immediately triggers a malicious login attempt using the already compromised victim’s Level 1 credentials.

Due to the protocol’s design, the legitimate notification on the user’s device is then overwritten by the attacker’s subsequent request, which leads the unsuspecting user, believing they are authorising their session, to inadvertently grant the attacker access to the account. This is a typical “race condition” exploit that highlights the danger of lacking unique

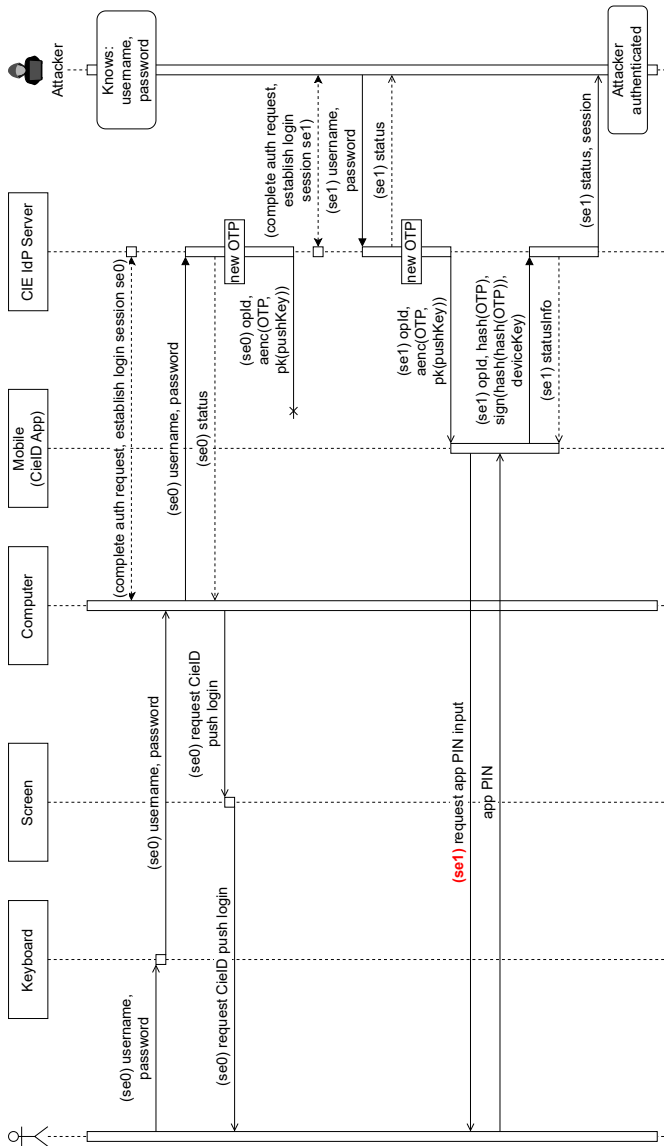


Figure 5.2: Overview of the L2PSH attack in the empty scenario.

session identifiers enabling users to verify the session correspondence or, at least, notification history in the mobile application.

The only discernible anomaly visible to the user is the momentary replacement of the legitimate notification with an identical malicious one: however, this transition is so subtle that it is unlikely to be detected by the vast majority of users, including those with high security awareness.

The feasibility of this exploit is not merely theoretical: it has been successfully demonstrated through a proof-of-concept implementation conducted on a real CIE account, confirming that the vulnerability can be weaponised in a real-world setting.

5.4.2 Level 3 protocols: L3APP, L3DSK

The CIE Level 3 protocols represent a substantial security advancement over Level 2, primarily through the integration of the physical CIE smart card as a hardware-based authentication factor.

To successfully compromise a Level 3 authentication flow, an adversary must either establish control over the communication channel with the physical smart card or employ social engineering to manipulate the user into performing cryptographic operations on its behalf. While the latter remains a viable threat vector, the requirement for physical interaction with a smart card is often perceived by users as a more unusual and potentially high-risk activity compared to purely digital interactions, introducing a psychological barrier that may make social engineering tactics appear more suspicious, increasing the likelihood of user intervention.

The results of our formal analysis regarding the CIE Level 3 protocols, including their resilience against various adversarial models, are summarised in Table 5.2.

L3APP The L3APP protocol utilises a substantial number of interfaces, resulting in a notably complex execution flow. This architectural complexity contributes to a broader attack surface than simpler protocols, as the transmission of sensitive data across multiple devices and interfaces creates diverse opportunities for exploitation at various layers.

Threat Scenario						L3APP	L3DSK					
					$\mathcal{A}_{io:RW}^{mob-sm}$	✗	—					
					$\mathcal{A}_{in:RW}^{mob-hum}$	✓	—					
					$\mathcal{A}_{in:RW}^{mob-hum}$	$\mathcal{A}_{io:RO}^{mob-tls}$	✗	—				
					$\mathcal{A}_{in:RW}^{mob-hum}$	$\mathcal{A}_{io:RO}^{mob-sm}$	✗	—				
					$\mathcal{A}_{io:RW}^{mob-hum}$		✗	—				
					$\mathcal{A}_{out:RO}^{pc-displ}$	$\mathcal{A}_{in:RW}^{mob-hum}$	✗	—				
					$\mathcal{A}_{out:RW}^{pc-displ}$		✗	✓				
					$\mathcal{A}_{in:RO}^{pc-kbd}$	$\mathcal{A}_{in:RW}^{mob-hum}$	✗	—				
					$\mathcal{A}_{in:RW}^{pc-kbd}$		✗	✗				
					$\mathcal{A}_{io:RO}^{pc-tls}$	$\mathcal{A}_{in:RW}^{mob-hum}$	✗	—				
					$\mathcal{A}_{io:RO}^{pc-tls}$	$\mathcal{A}_{in:RO}^{pc-kbd}$	$\mathcal{A}_{out:RO}^{pc-displ}$	$\mathcal{A}_{io:RO}^{mob-hum}$	$\mathcal{A}_{io:RO}^{mob-sm}$	$\mathcal{A}_{io:RW}^{mob-tls}$	✓	—
					$\mathcal{A}_{io:RW}^{pc-tls}$						✗	✓
					$\mathcal{A}_{io:RW}^{pc-sm}$						—	✗
PH											✗	✓
PH	$\mathcal{A}_{io:RO}^{pc-sm}$	$\mathcal{A}_{io:RW}^{pc-tls}$	$\mathcal{A}_{in:RO}^{pc-kbd}$	$\mathcal{A}_{out:RW}^{pc-displ}$							—	✓

Table 5.2: Analysis results of Level 3 CIE protocols under different threat scenarios. See Table 5.1 for the legend.

The impact of this complexity is illustrated in Table 5.2, which reports a significant number of proven attacks: these findings represent distinct, independent attack paths discovered by ProVerif: each path corresponds to a specific combination of interface compromises that allows an adversary to successfully bypass the Level 3 security guarantees.

In practice, many of the identified vulnerabilities focus on the concurrent recovery of the PIN and the OTP through various interception techniques. For example, if we consider an adversary capable of initiating communication with the smart card to force a signature of the IdP challenge (modelled as having control over the *mob-hum* interface input stream), we realise that this capability alone does not suffice for authentication. While the attacker can acquire the smart card PIN by observing user input during a legitimate session, they must still obtain the resulting OTP to proceed with the protocol execution. Our analysis identifies several distinct paths for this secondary acquisition:

- by gaining full read-write access to the user’s mobile device, an

attacker can directly observe the screen and capture the displayed OTP ($\{\mathcal{A}_{io:RW}^{mob-hum}\}$);

- an attacker can record the OTP as the user manually enters it into the computer by logging keystrokes on the computer’s keyboard ($\{\mathcal{A}_{in:RO}^{pc-kbd}, \mathcal{A}_{in:RW}^{mob-hum}\}$);
- the OTP can be intercepted by compromising the TLS interface between the computer and the IdP server, allowing the attacker to read the sensitive credential during transmission ($\{\mathcal{A}_{io:RO}^{pc-tls}, \mathcal{A}_{in:RW}^{mob-hum}\}$).

Moreover, as with the L2SMS protocol, the OTP is displayed on the computer’s screen, rendering it observable by an adversary with control over the display output channel. This vulnerability can be exploited by malware specifically designed to capture screen output.

Unfortunately, the structural complexity involved in the handling and transmission of secrets within L3APP suggests that the identified vulnerabilities cannot be solved through minor adjustments alone. Despite these challenges, L3APP maintains a security level at least equivalent to L2QRC, with the added benefit of hardware-based verification.

In fact, the protocol’s superior resilience is largely derived from the requirement for the physical smart card to be in direct proximity to the mobile device during authentication: the necessity for a physical token introduces a deliberate user interaction that is likely to evoke inherent suspicion in a malicious context, thereby increasing the difficulty for attackers to successfully employ social engineering tactics.

L3DSK The L3DSK protocol distinguishes itself as the most robust within the CIE suite, providing formal guarantees even in scenarios where phishing is feasible and the adversary gains access to multiple system interfaces, as indicated in the final row of Table 5.2.

Despite its resilience, our analysis identifies two distinct failure modes. The first occurs when the communication channel between the computer and the smart card is compromised with read-write access ($\{\mathcal{A}_{io:RW}^{pc-sm}\}$). This vulnerability is anticipated, as such a degree of control allows an

attacker to bypass or emulate the operating system’s smart card middleware entirely. The second vulnerability involves an attacker successfully capturing the card’s PIN and subsequently replaying or re-entering it to authorise a rogue session by manipulating the computer’s keyboard channel ($\{\mathcal{A}_{\text{in:RW}}^{\text{pc-kbd}}\}$). These limitations appear to be intrinsic to the protocol’s design and are unlikely to be mitigated through minor adjustments alone.

Mitigating these security concerns, particularly the PIN retrieval and replay attack, necessitates a radical architectural change: one potential solution involves replacing the standard smart card with a personal, tamper-resistant hardware authenticator. This dedicated device could feature an embedded screen to display session-specific information, such as a dynamic challenge, thereby enhancing user awareness of the transaction context. Furthermore, an integrated, tamper-proof keypad would provide a secure channel for PIN entry, effectively neutralising the risk of software-based eavesdropping on the computer. While such a hardware-centric approach may be currently impractical for broad eIDAS implementations due to established standards and the high cost of infrastructure, it offers the enhanced security guarantees required for smaller, controlled, high-security environments where dedicated hardware is feasible.

5.5 On the scope of symbolic modelling and analysis

Our framework treats cryptographic primitives as “perfect black boxes,” adhering to the Dolev-Yao algebraic assumptions: under this symbolic model, we assume that an adversary cannot compromise the cryptography itself, e.g. they cannot guess a nonce or invert a hash function, but can only perform operations for which they possess the necessary keys.

This methodology intentionally abstracts away the computational complexities and implementation-level vulnerabilities of specific attack vectors, such as side-channel analysis, differential cryptanalysis, or brute-force attempts. By neutralising these variables, we can focus exclusively on a rigorous formal verification of the protocol’s structural logic. This allows us to identify fundamental design flaws, such as improper session

binding or insecure message sequencing, which would remain critical regardless of the cryptographic algorithms’ strength.

The alignment between symbolic abstractions and real-world deployments is a foundational challenge in formal methods (Abadi et al. 2002; Blanchet 2008; Wang et al. 2014). As with any scientific model, a fundamental trade-off must be managed: excessive abstraction risks overlooking implementation-specific vulnerabilities; on the other hand, excessive granularity often renders systematic reasoning computationally infeasible due to the state-space explosion.

We position the symbolic model as a *functional specification* that defines the mandatory security properties any compliant implementation must satisfy. By establishing this baseline, we decouple the protocol’s logical integrity from the specific performance of its cryptographic components.

For instance, if a protocol is formally proven secure under the assumption of perfect encryption, the subsequent use of a weak or compromised primitive in a real-world deployment represents a failure of the implementation to meet the specification, rather than a flaw in the model itself. This abstraction allows for a modular security analysis: the symbolic model ensures the structural correctness of the message exchange, while the computational implementation must independently ensure the strength of the primitives chosen to instantiate that structure. A direct consequence of this is that the two layers may evolve independently: as long as a replacement primitive preserves the properties assumed by the symbolic model, e.g. the secrecy of keys or the collision resistance of a hash function, it can be substituted without invalidating the existing security proof.

We believe that this approach is particularly valuable for network protocols and standards like eIDAS, as it provides a platform-independent benchmark for security. It allows developers and auditors to distinguish between vulnerabilities arising from “how the protocol is designed” versus “how the code is written,” ensuring that the core logic is sound before addressing the complexities of the underlying cryptographic libraries.

While a model’s primary role is to provide a formal abstraction, it must be refined if it fails to capture a relevant class of threats. Our interface-based methodology is designed specifically to enable this evolution: its

modular and extensible architecture allows for the integration of new components or interfaces. This flexibility also enables the formalisation of emerging threat vectors (such as side-channel attacks, fault injection, or hardware-level exploits) as they are identified, ensuring the model remains robust and relevant in an evolving adversarial landscape.

This approach establishes a clear hierarchy of security: a symbolic proof is a necessary, though not exhaustive, precondition for both computational and physical security. A protocol that is flawed at the symbolic level is fundamentally broken: no cryptographic implementation, regardless of its strength, can “rescue” an inherently insecure architecture.

Consequently, following the “Fail-Fast” principle, our framework acts as a critical primary filter. It allows for the early identification of protocol architectures that are structurally robust against defined threat models, ensuring that only logically sound designs proceed to the more resource-intensive stages of cryptographic verification and hardware hardening.

This analysis is intended to be *complementary* to heuristic and computational methodologies (Cortier et al. 2011; Wang et al. 2014). The primary strength of our framework lies in its systematic exhaustiveness: by abstracting the mechanics of an attack into its logical consequences we can automatically evaluate thousands of adversarial scenarios in a matter of hours. Such a comprehensive exploration of the state space would otherwise require years of manual heuristic effort and would remain significantly more prone to human error.

By establishing this baseline, we provide an architectural blueprint for authentication protocols (and eIDAS-compliant Multi-Factor Authentication in particular). Once a protocol is symbolically verified, heuristic and computational analyses can be targeted toward the high-value task of bridging the “last mile” between these verified specifications and their physical implementations and deployment.

This approach ensures that implementation-specific details satisfy the high-assurance requirements of the eIDAS ecosystem. By filtering out structural flaws at the design stage, we allow security auditors to concentrate their resources on verifying that the actual code and hardware execution strictly adhere to the logically sound blueprint.

5.6 Protocols improvement

The analysis detailed in Section 5.4 demonstrates that all Level 2 and Level 3 protocols exhibit some vulnerabilities when subjected to specific adversarial scenarios. For instance, in L2SMS, an attacker who gains control over the user's telephony provider, such as through a SIM swap attack, can intercept the SMS-delivered passcode. Similarly, in L2PSH, if the smartphone's execution environment is compromised by malware, an adversary could potentially hijack or spoof push notifications to authorise fraudulent sessions. In the case of L2QRC, an attacker with the ability to manipulate the computer's display could present a malicious QR code, misleading the user into authenticating a rogue transaction. These examples show the inherent weaknesses present in each protocol: while they provide varying degrees of protection, each remains susceptible to compromise once specific trust assumptions regarding the underlying hardware or communication channels are violated.

It is important to note that the majority of these vulnerabilities arise from attacks targeting the communication channels and the end-user devices themselves, rather than originating from inherent logical flaws within the protocol specifications. Consequently, mitigating these risks would likely require a comprehensive redesign of the underlying system architecture rather than minor protocol patches.

Nevertheless, by incorporating these attack vectors into our threat model, we can systematically analyse the protocols' resilience even in compromised environments. Our analysis demonstrates that certain designs exhibit superior isolation; for instance, L2QRC maintains its security guarantees even if TLS traffic is decrypted or keyboard input is intercepted. This suggests that while the platform may be untrusted, the structural binding of the protocol shields the authentication secret from the host's vulnerabilities. Similarly, L3DSK remains secure even when phishing is possible and the attacker gains access to multiple interfaces.

Still, certain weaknesses are intrinsic to the protocol designs themselves and can be mitigated through targeted, minor modifications. We propose enhancements for the L2SMS, L3APP, and L2PSH protocols to

resolve the display channel vulnerability inherent in the former two and the “notification overwrite” vulnerability present in the latter.

5.6.1 L2SMS*: L2SMS improvement

The ProVerif counter-example trace shows that the vulnerability in L2SMS originates from the transmission of the OTP to the computer’s display. By rendering the secret in plaintext on the user’s screen, the protocol inadvertently exposes the credential to any adversary with read-only access to the display output channel.

This exposure facilitates exploitation through common remote-access vectors, such as unauthorised screen sharing or malware-based screen capture, as well as physical threats like “shoulder surfing.” Since the computer screen is treated as a trusted output in the original design, the compromise of this single channel effectively nullifies the secrecy of the secondary authentication factor.

To rectify this issue, it is sufficient to eliminate the displaying of the OTP to the computer’s display. By ensuring the secret is never rendered on the screen, we remove the vector for display-based interception. Formal verification of this modified protocol, denoted as L2SMS*, confirms the successful mitigation of this vulnerability (see Table 5.3).

Our analysis indicates that an adversary possessing either read-only or read-write access to the display channel is no longer able to achieve unauthorised access. The verification of the revised protocol reveals no further security concerns within the defined threat model, demonstrating that the structural change effectively isolates the authentication secret from the compromised output channel.

In practice, to prevent an adversary from observing the OTP during entry, the protocol could provide no visual feedback at all, effectively suppressing any character display on the screen while the user types. Alternatively, characters can be masked by replacing the actual digits with asterisks or bullets as they are entered. While this approach provides basic user confirmation that the keystrokes are being registered, it still successfully shields the actual digits from screen-based interception. Both

Threat Scenario						L2SMS*	L2SMS*	L2PSH*	L2PSH*	L2QRC	L3APP	L3APP*	L3DSK
						✓	✓	✗	✓	✓	✓	✓	✓
$\mathcal{A}^{\text{mob-sm}}$ in:RO						✗	✗	—	—	—	—	—	—
$\mathcal{A}^{\text{mob-sm}}$ io:RW						—	—	—	—	—	—	—	—
						$\mathcal{A}^{\text{mob-hum}}$ in:RW	$\mathcal{A}^{\text{mob-tls}}$ io:RO	✓	✓	✗	(✗)	✗	✗
						$\mathcal{A}^{\text{mob-hum}}$ in:RW	—	—	✗	✗	✗	✗	—
$\mathcal{A}^{\text{mob-sm}}$ io:RO						$\mathcal{A}^{\text{mob-hum}}$ in:RW	—	—	—	—	✗	✗	—
						$\mathcal{A}^{\text{mob-hum}}$ io:RO	—	—	—	—	✗	✗	—
						$\mathcal{A}^{\text{mob-hum}}$ io:RW	✗	✗	✗	✓	✓	✓	—
						$\mathcal{A}^{\text{mob-hum}}$ io:RW	✗	✗	✗	✗	✗	✗	—
						$\mathcal{A}^{\text{pc-displ}}$ out:RO	✗	✓	✗	✓	✓	✓	✓
						$\mathcal{A}^{\text{pc-displ}}$ out:RO	✗	✓	✗	✗	✗	✗	—
						$\mathcal{A}^{\text{pc-displ}}$ out:RW	✗	✓	✗	✗	✗	✗	✓
						$\mathcal{A}^{\text{pc-displ}}$ out:RW	—	—	✗	✗	✗	✗	—
$\mathcal{A}^{\text{mob-sm}}$ io:RO						$\mathcal{A}^{\text{mob-tls}}$ io:RO	—	—	✗	✗	✗	✗	—
						$\mathcal{A}^{\text{mob-hum}}$ in:RW	—	—	—	—	✗	✗	—
						$\mathcal{A}^{\text{mob-hum}}$ io:RW	✗	✓	✗	✗	✗	✓	—
						$\mathcal{A}^{\text{mob-hum}}$ io:RO	✗	✗	✗	✗	✗	✗	—
						$\mathcal{A}^{\text{pc-kbd}}$ in:RO	✗	✗	✗	✓	✓	✓	✓
						$\mathcal{A}^{\text{pc-kbd}}$ in:RO	✗	✗	✗	✗	✗	✗	—
						$\mathcal{A}^{\text{pc-kbd}}$ in:RO	✗	✗	✗	✗	✗	✗	—
						$\mathcal{A}^{\text{pc-kbd}}$ in:RO	✗	✗	✗	✗	✗	✗	✓
						$\mathcal{A}^{\text{pc-kbd}}$ in:RW	✗	✗	✗	✗	✗	✗	✗
						$\mathcal{A}^{\text{pc-tls}}$ io:RO	✗	✗	✗	✓	✓	✓	—
						$\mathcal{A}^{\text{pc-tls}}$ io:RO	✗	✗	✗	✗	✗	✗	—
						$\mathcal{A}^{\text{pc-tls}}$ io:RO	✗	✗	✗	✗	✗	✗	—
						$\mathcal{A}^{\text{pc-tls}}$ io:RO	✗	✗	✗	✗	✗	✗	—
						$\mathcal{A}^{\text{pc-kbd}}$ in:RO	—	—	✗	✓	✓	—	—
						$\mathcal{A}^{\text{pc-kbd}}$ in:RO	—	—	✗	✓	—	—	—
						$\mathcal{A}^{\text{pc-kbd}}$ in:RO	—	—	✗	✓	—	—	—
						$\mathcal{A}^{\text{pc-kbd}}$ in:RO	—	—	—	—	✓	✓	—
						$\mathcal{A}^{\text{pc-kbd}}$ in:RO	—	—	—	—	✓	✓	—
						$\mathcal{A}^{\text{pc-kbd}}$ in:RO	—	—	—	—	✓	✓	—
						$\mathcal{A}^{\text{pc-kbd}}$ in:RO	—	—	—	—	✓	✓	—
						$\mathcal{A}^{\text{pc-kbd}}$ in:RO	—	—	—	—	✓	✓	—
						$\mathcal{A}^{\text{pc-kbd}}$ in:RO	—	—	—	—	✓	✓	—
						$\mathcal{A}^{\text{pc-kbd}}$ in:RO	—	—	—	—	✓	✓	—
						$\mathcal{A}^{\text{pc-kbd}}$ in:RO	—	—	—	—	✓	✓	—
						$\mathcal{A}^{\text{pc-kbd}}$ in:RO	—	—	—	—	✓	✓	—
						$\mathcal{A}^{\text{pc-kbd}}$ in:RO	—	—	—	—	✓	✓	—
						$\mathcal{A}^{\text{pc-kbd}}$ in:RO	—	—	—	—	✓	✓	—
						$\mathcal{A}^{\text{pc-kbd}}$ in:RO	—	—	—	—	✓	✓	—
						$\mathcal{A}^{\text{pc-kbd}}$ in:RO	—	—	—	—	✓	✓	—
						$\mathcal{A}^{\text{pc-kbd}}$ in:RO	—	—	—	—	✓	✓	—
						$\mathcal{A}^{\text{pc-kbd}}$ in:RO	—	—	—	—	✓	✓	—
						$\mathcal{A}^{\text{pc-kbd}}$ in:RO	—	—	—	—	✓	✓	—
						$\mathcal{A}^{\text{pc-kbd}}$ in:RO	—	—	—	—	✓	✓	—
						$\mathcal{A}^{\text{pc-kbd}}$ in:RO	—	—	—	—	✓	✓	—
						$\mathcal{A}^{\text{pc-kbd}}$ in:RO	—	—	—	—	✓	✓	—
						$\mathcal{A}^{\text{pc-kbd}}$ in:RO	—	—	—	—	✓	✓	—
						$\mathcal{A}^{\text{pc-kbd}}$ in:RO	—	—	—	—	✓	✓	—
						$\mathcal{A}^{\text{pc-kbd}}$ in:RO	—	—	—	—	✓	✓	—
						$\mathcal{A}^{\text{pc-kbd}}$ in:RO	—	—	—	—	✓	✓	—
						$\mathcal{A}^{\text{pc-kbd}}$ in:RO	—	—	—	—	✓	✓	—
						$\mathcal{A}^{\text{pc-kbd}}$ in:RO	—	—	—	—	✓	✓	—
						$\mathcal{A}^{\text{pc-kbd}}$ in:RO	—	—	—	—	✓	✓	—
						$\mathcal{A}^{\text{pc-kbd}}$ in:RO	—	—	—	—	✓	✓	—
						$\mathcal{A}^{\text{pc-kbd}}$ in:RO	—	—	—	—	✓	✓	—
						$\mathcal{A}^{\text{pc-kbd}}$ in:RO	—	—	—	—	✓	✓	—
						$\mathcal{A}^{\text{pc-kbd}}$ in:RO	—	—	—	—	✓	✓	—
						$\mathcal{A}^{\text{pc-kbd}}$ in:RO	—	—	—	—	✓	✓	—
						$\mathcal{A}^{\text{pc-kbd}}$ in:RO	—	—	—	—	✓	✓	—
						$\mathcal{A}^{\text{pc-kbd}}$ in:RO	—	—	—	—	✓	✓	—
						$\mathcal{A}^{\text{pc-kbd}}$ in:RO	—	—	—	—	✓	✓	—
						$\mathcal{A}^{\text{pc-kbd}}$ in:RO	—	—	—	—	✓	✓	—
						$\mathcal{A}^{\text{pc-kbd}}$ in:RO	—	—	—	—	✓	✓	—
						$\mathcal{A}^{\text{pc-kbd}}$ in:RO	—	—	—	—	✓	✓	—
						$\mathcal{A}^{\text{pc-kbd}}$ in:RO	—	—	—	—	✓	✓	—
						$\mathcal{A}^{\text{pc-kbd}}$ in:RO	—	—	—	—	✓	✓	—
						$\mathcal{A}^{\text{pc-kbd}}$ in:RO	—	—	—	—	✓	✓	—
						$\mathcal{A}^{\text{pc-kbd}}$ in:RO	—	—	—	—	✓	✓	—
						$\mathcal{A}^{\text{pc-kbd}}$ in:RO	—	—	—	—	✓	✓	—
						$\mathcal{A}^{\text{pc-kbd}}$ in:RO	—	—	—	—	✓	✓	—
						$\mathcal{A}^{\text{pc-kbd}}$ in:RO	—	—	—	—	✓	✓	—
						$\mathcal{A}^{\text{pc-kbd}}$ in:RO	—	—	—	—	✓	✓	—
						$\mathcal{A}^{\text{pc-kbd}}$ in:RO	—	—	—	—	✓	✓	—
						$\mathcal{A}$							

Results:

- (✗) ProVerif finds attack after a minor manual modification to the scenario's code.

Scenarios:

- NC: no-check scenario (the user skips the verification of the session identifier)

Table 5.3: Results comparison with the improved CIE protocols. See Table 5.1 for the rest of legend.

methods ensure that an attacker with read access to the display channel cannot reconstruct the secret, though the masked approach is generally preferred for usability as it reduces user error during the flow.

5.6.2 L3APP*: L3APP improvement

Our analysis of L3APP identifies a vulnerability analogous to the one discovered in L2SMS: both protocols render the OTP in plaintext on the computer screen, exposing it to any adversary able to gain read access to the display channel. To remediate this, we propose L3APP*, an enhancement mirroring our approach for L2SMS*: masking the OTP characters in real-time as the user inputs them. This measure ensures that even an attacker with active screen-monitoring capabilities is prevented from observing or capturing the sensitive credential.

Formal verification of L3APP* confirms the successful mitigation of the vulnerability. The results in Table 5.3 demonstrate that the protocol remains secure even in scenarios where the adversary maintains full control over the display channel.

5.6.3 L2PSH*: L2PSH improvement

The attack trace visualised in Figure 5.2 exposes a critical vulnerability intrinsic to the protocol’s design. The ProVerif analysis demonstrates that this flaw can be exploited even in the uncompromised system, i.e. where no specific device interfaces have been attacked.

This finding indicates that the vulnerability is not due to external environmental failures, but rather a fundamental logical error in the protocol’s message sequencing. In such cases, the adversary is able to reach the goal (i.e. invalidate the ProVerif query) simply by interacting with the protocol as specified, leveraging legitimate messages to induce an insecure state.

To address this vulnerability and strengthen the protocol’s security, we propose a verification mechanism for push notifications that enables users to confirm that the notification received on their mobile device is logically bound to the specific login attempt they initiated on their computer.

Under this refined logic, if the verification parameters (such as a session-specific matching code) do not align, the user must terminate the authentication flow to prevent unauthorised access. We denote this enhanced protocol as L2PSH*, as illustrated in Figure 5.3. By introducing this explicit binding, we eliminate the risk of notification injection attacks, where the adversary attempts to piggyback on a user’s active session or trick them into approving their rogue request.

Specifically, the modified protocol introduces a mutual verification step to ensure session integrity. During the login process, a unique identifier, such as a numeric code or a short passphrase, is generated by the IdP server and rendered on the user’s computer screen. Simultaneously, this same identifier is embedded within the push notification delivered to the user’s mobile device.

Before authorising the request within the CieID mobile app, the user must visually verify that the two codes are identical. This manual comparison creates an out-of-band validation check: the user only proceeds if the mobile prompt explicitly matches the computer session they initiated. By introducing this binding, the protocol ensures that an attacker cannot trick the user into approving a malicious authentication request.

While the proposed session code mechanism is theoretically robust, its practical efficacy is limited by the risk of user fatigue or human error, as users may habitually approve requests without performing the necessary verification. To mitigate this “autopilot” behaviour, we recommend a selection-based challenge to ensure active engagement.

In this model, the user is not a passive observer: they must identify and select the correct matching code from a randomised grid displayed on their mobile device. This interaction mandates a cognitive check, forcing the user to consciously confirm the link between the computer session and the mobile authorisation before the authentication can proceed.

By transitioning from a passive (accept/deny) to an active selection task, the protocol becomes significantly harder to exploit by adversarial social engineering. Even if an attacker triggers a rogue notification, the user cannot inadvertently authorise it without possessing the specific session-bound value displayed on the illegitimate terminal.

As demonstrated in Table 5.3, the formal analysis in ProVerif confirms that the enhancements in L2PSH* successfully mitigate the vulnerabilities in the original L2PSH design: by introducing the session-binding mechanism, the protocol’s security profile is elevated to a level equivalent to that of L2QRC. This parity is significant because it indicates that both protocols now provide the same degree of resistance against unauthorised access, even when the communication channels are partially compromised.

The verification of L2PSH* in the $\{\mathcal{A}_{\text{in.RW}}^{\text{mob-hum}}\}$ scenario (representing an attacker with read-write access to the mobile device’s human-machine interface) warrants a detailed discussion. Among the hundreds of scenarios automatically generated by the algorithm in Figure 5.1, this is the only case where ProVerif terminates with the result “Cannot be proven.”

As noted in Section 5.3, this result occurs when the tool’s resolution algorithm identifies a logical derivation of an attack but is unable to reconstruct a corresponding, valid execution trace. This often originates from the over-approximations inherent in the Horn clause model: while a mathematical path to the secret exists in the abstract logic, it may violate temporal or state-based constraints when mapped back to the actual protocol flow. In this specific scenario, the tool suggests that an attacker could theoretically intercept the session code and bypass the user’s manual check, yet it cannot find a sequence of actions that makes this physically possible given the protocol’s structure.

This “Cannot be proven” status represents a limitation of the symbolic abstraction rather than a confirmed vulnerability. It indicates that the security property is too wide for the automated tool to resolve definitively without further guidance or more granular lemmas. Consequently, a formal symbolic proof cannot be claimed for this specific corner case. The absence of a reconstructible attack trace is ambiguous: it may suggest that the protocol is sound in practice, or (more likely) that a vulnerability exists but the tool is unable to translate it into a concrete attack.

In fact, the attack is authentic. This is confirmed by a manual inspection of ProVerif’s logical derivation. Furthermore, by applying targeted model refinements, we can assist the tool in successfully reconstructing a concrete attack trace, thereby providing formal confirmation of the threat.

The applied minor refinement to the model to assist the reconstruction algorithm consists, specifically, of introducing parallelism immediately preceding the OTP output on the mobile screen.

This modification effectively grants the attacker temporary Write access to the mobile screen. Although this represents an over-approximation of our intended threat model, as the attacker should not technically possess write access to the screen in this specific case, it serves as a catalyst for ProVerif’s search process. In this extended model, ProVerif successfully reconstructs the attack trace, confirming the feasibility of the threat.

The attack involves retrieving the app PIN by exploiting read access to the mobile device’s touchscreen and subsequently utilising that PIN in a separate, concurrent session to authorise a login attempt initiated by the adversary on its own device.

A high-level visualisation of this semi-automatically discovered attack trace is provided in Figure 5.4.

Crucially, this attack trace *does not* exploit the artificially introduced $\{\mathcal{A}_{\text{out:RW}}^{\text{mob-hum}}\}$ capability; the attacker performs no write operations on the mobile device’s screen. This demonstrates that the attack is feasible within the more restrictive, original scenario $\{\mathcal{A}_{\text{in:RW}}^{\text{mob-hum}}\}$. Consequently, we can confirm that the initial “Cannot be proven” result was indeed masking a genuine protocol flaw rather than being a false positive generated by the tool’s abstractions.

5.7 Resource utilization of the CIE analysis

To evaluate the practical computational feasibility and scalability of the CIE analysis, we conduct a brief resource utilization analysis across all evaluated protocol scenarios.

All benchmark experiments are executed on a computer equipped with an Intel(R) Core(TM) i7-8750H (12 thread) @ 4.10 GHz and 32 GB of DDR4 RAM. The host operating system utilised is Arch Linux running the Linux kernel version 7.0.12-arch1-1.

To ensure empirical fidelity and isolate system noise, execution time and maximum resident memory metrics are gathered using the standard

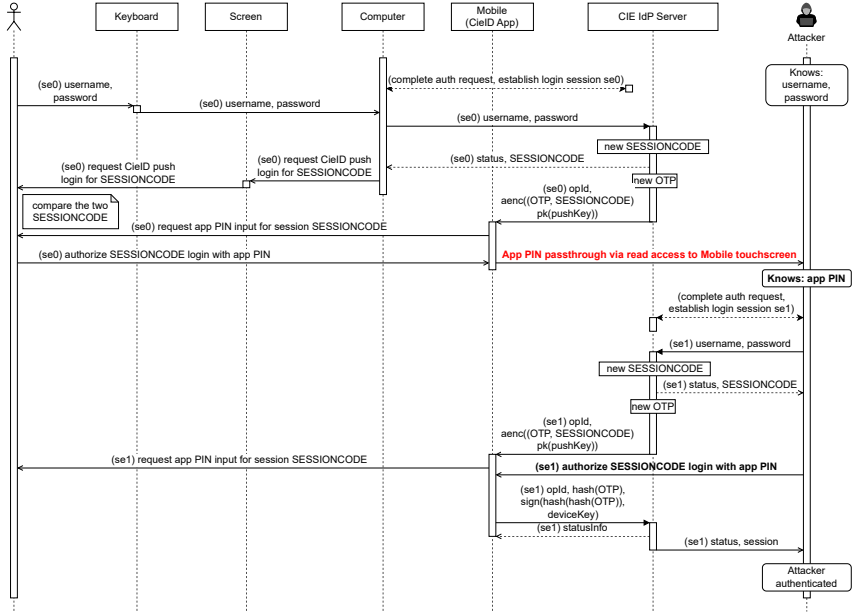


Figure 5.4: Overview of the attack to L2PSH* in the $\{\mathcal{A}_{in:RW}^{mob-hum}\}$ scenario.

GNU `time` utility. Each individual protocol verification benchmark is independently repeated over 10 consecutive iterations. From this sample space, the empirical arithmetic mean (Avg.) and sample standard deviation (σ) are computed. The aggregated experimental results are presented in Table 5.4 and visualised in Figure 5.5.

Despite using a consumer-grade computer that does not represent modern professional-grade hardware, the formal verification process across the majority of evaluated protocols concluded rapidly. For almost all Level 2 protocols (L2SMS, L2SMS*, L2PSH, L2QRC) along with the Level 3 Desktop protocol (L3DSK), the verification framework consistently yielded results in sub-second execution windows.

Specifically, execution times for these configurations ranged from a minimum average of 0.0474 ± 0.0069 s (L2PSH) to 0.1940 ± 0.0181 s (L3DSK). Correspondingly, memory consumption remained remarkably flat and bounded within a modest footprint of approximately 26.9 MB to 29.3 MB.

Protocol	Execution time (s)		Memory usage (kB)	
	Avg.	σ	Avg.	σ
L2SMS	0.0673	0.0031	27 198.8	134.73
L2SMS*	0.0948	0.0051	27 212.0	149.81
L2PSH	0.0474	0.0069	27 946.4	88.65
L2PSH*	2.8962	0.2038	29 302.0	144.89
L2QRC	0.1892	0.0059	27 132.4	186.73
L3APP	267.7887	13.2994	415 356.8	66 787.20
L3APP*	248.8493	8.9764	552 502.0	37 490.80
L3DSK	0.1940	0.0181	26 921.6	83.85

Table 5.4: Performance metrics (execution time and memory consumption) for different CIE Level 2 and 3 protocol verifications.

For L2PSH*, execution time extended to 2.8962 ± 0.2038 s: this represents an explicit, yet manageable, increase in state-space exploration requirements without a proportional expansion in peak memory, which stabilised at 29.3 MB.

The notable anomaly within this empirical evaluation lies in the Level 3 App variants, namely L3APP and its derivative L3APP*. As the most convoluted protocols under evaluation, their underlying state-space complexity scales due to an increased number of interacting processes and messages. This behavioural shift is evident when analysing the visual distributions in Figure 5.5.

For the L3APP protocol, the execution time rises to 267.7887 ± 13.2994 s, accompanied by a heavy RAM footprint averaging 415.356 MB. Interestingly, its modified counterpart, L3APP*, exhibits a slight optimization in computational throughput, resolving in a lower average duration of 248.8493 s. However, this marginal time optimization is achieved at the expense of a larger spatial complexity: the maximum resident set size for L3APP* reaches $552\,502.0 \pm 37\,490.80$ kB (more than half a GB).

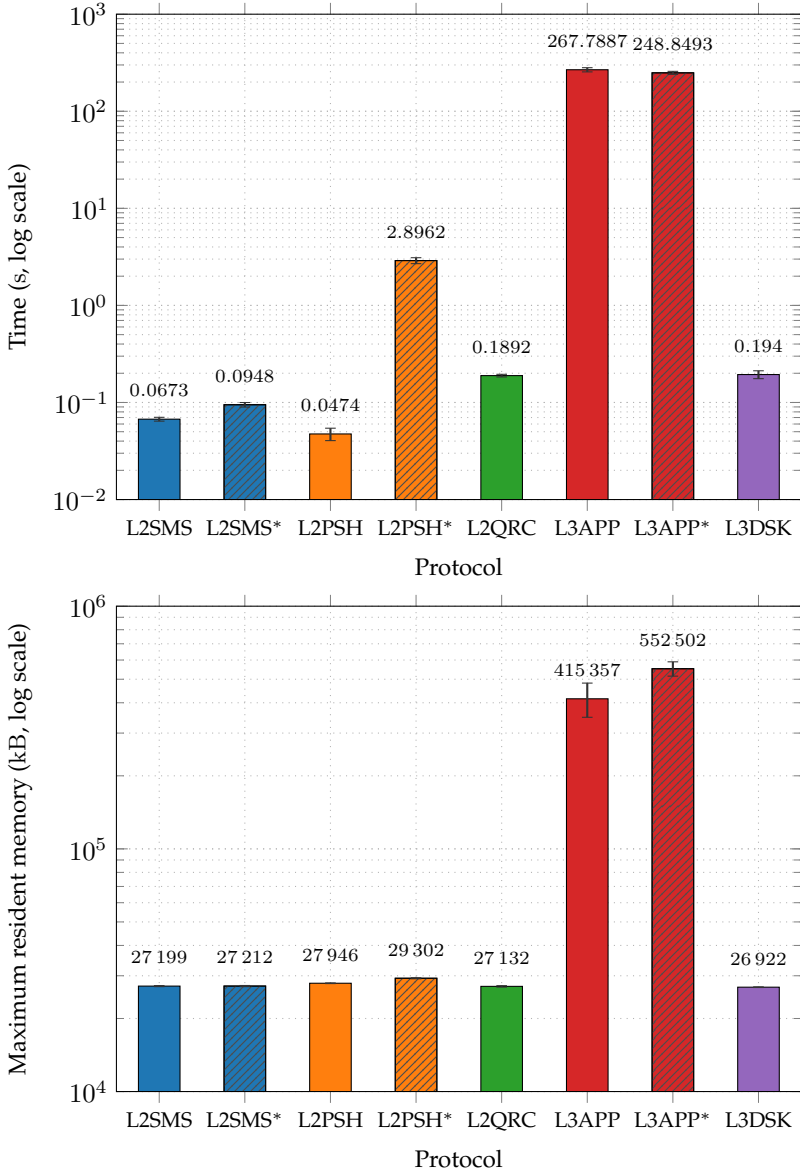


Figure 5.5: Computational resource footprints across CIE protocols: execution time and maximum resident memory usage. Log scale.

Chapter 6

Extending interface-based models to decoupled systems

Security analysis has historically relied on the assumption of spatiotemporal coupling. In this traditional view, two entities exchange messages over a dedicated channel at a specific moment in time. However, the explosion of the *Internet of Things* (IoT) and *Industrial IoT* (IIoT) has rendered this model insufficient. To accurately verify these systems, we must evolve our formalisms to account for communication that is distributed, asynchronous, and decoupled.

In many IoT ecosystems, the sender and receiver may never interact directly. We categorise the limitations of traditional models into two areas:

Spatial Decoupling when the sender does not need to know the identity or address of the receivers. Often even the receiver is unaware of the sender's identity. Messages are sent to the “network” rather than a specific peer;

Temporal Decoupling if the parties do not need to be active at the same time. A sensor might publish data to a broker, which stores it until a controller wakes up to process it.

While our previous analysis of protocols focused on direct interactions, IoT systems often rely on a broadcast or multicast nature that breaks these one-to-one assumptions. When communication is coupled, “attacking an interface” is conceptually simple: the adversary targets the link between A and B. In a decoupled environment (like a wireless sensor network or a shared industrial bus), the definition of an interface attack becomes far more complex.

For example, if an attacker injects a malicious packet onto a broadcast interface the message is not directed at one single victim but it is rather processed by every participant within the broadcast domain, thus enabling a single unauthorised command to trigger a potential state change across an entire fleet of devices. Moreover, in a multicast group, verifying the true origin of a message may become a challenge, as the shared nature of the communication often masks individual sender identities.

To capture these dynamics formally, we extend our verification framework to include the *publish/subscribe* (pub/sub) paradigm. This is the architectural standard for protocols like MQTT (formerly *Message Queuing Telemetry Transport*) or *Advanced Message Queuing Protocol* (AMQP), and the *Data Distribution Service* (DDS) standard.

By modelling pub/sub, we can represent various communication patterns as abstractions of topics: for example, broadcast can be represented with a “global topic” where every node in the system is a subscriber; or multicast with specialised topics where only authorised “groups” are enabled to subscribe or publish. The model also takes into account the fact that publishers and subscribers are isolated by the broker (acting like a shared bus), potentially introducing new attack vectors.

It is important to notice that in this chapter we extend, rather than replace, the existing coupled model introduced in Chapter 4. This is a necessity because modern distributed systems rarely rely on a single communication paradigm. Consider the Robot Operating System (ROS), the industry standard for robotics. It possesses both kinds of interactions and in real systems they usually operate together:

- Uncoupled (DDS Topics): Robots use pub/sub to broadcast continuous telemetry, such as battery status or GPS position;

- Coupled (Services/ Actions): When a central controller needs a robot to perform a specific task (e.g. “Open Valve 4”), it typically uses a direct, point-to-point service call.

A vulnerability in the pub/sub telemetry (e.g. spoofing a “low battery” signal) might be used to force the system into a state where a direct service call becomes exploitable. Our extended model allows for the formal verification of these cross-paradigm attacks.

The need for a hybrid approach is further validated by classical distributed computing. Algorithms for Atomic Broadcast (ABCAST), such as those found in Birman et al. (1987), also rely on a mix of communication styles: multicast to disseminate information efficiently to a group; and point-to-point messages for the acknowledgement and “token-passing” phases to ensure all nodes agree on the order of messages.

Failure to model both the multicast and the point-to-point components would result in an incomplete security proof. By formalising the pub/sub layer alongside direct interfaces, we can finally verify properties like ordering and integrity of these distributed algorithms.

6.1 Pub/sub Dolev-Yao model

The classical Dolev-Yao model (Dolev et al. 1983), which assumes a point-to-point attacker-in-the-middle between two fixed endpoints, is inadequate for capturing the security aspects of decoupled communication. In a pub/sub context, we can no longer assume that a session is restricted to two participants, nor that controlling the communication channel implies control over the entire distributed data flow.

We thus introduce a *pub/sub Dolev-Yao model*, visualised in Figure 6.1. This model extends the adversary’s capabilities to account for the many-to-many nature of decoupled communication.

While our model retains the capabilities of a classical Dolev-Yao adversary, such as the power to intercept, delete, or inject messages, it refines the scope of these actions by introducing a more granular control over the network fabric. In a decoupled environment, the attacker is no longer

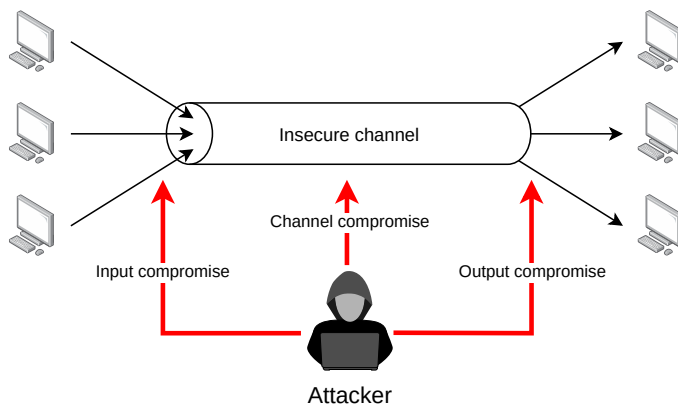


Figure 6.1: Pub/sub Dolev-Yao model.

perceived as a monolithic presence sitting on the single wire: instead, their influence is modelled as a collection of specific permissions and targeted compromises at the interface level. This allows for the formalisation of complex scenarios where an adversary might compromise the central broker entirely to gain visibility into routing metadata and unencrypted traffic, or conversely, only gain access to specific inputs or outputs of the communication system.

The flexibility of this approach enables the modelling of asymmetric privileges that are common in modern messaging architectures. For example, an attacker may possess the credentials necessary to publish to a specific topic, thereby injecting data into the system, yet remain unable to subscribe to that same topic to read the contents of messages sent by others. Alternatively, a curious broker might be modelled with the ability to read all passing content while lacking the cryptographic keys required to forge valid publications. This granularity is critical for analysing the security of pub/sub systems where different administrative domains often interact, as it allows to model systems that remain resilient even when specific data flows are under the attacker's control. Actually, the total compromise of a communication channel or broker can be conceptualised as the case where every input and output interface is simultaneously

compromised, granting the adversary complete control over the data flow. By framing the threat model in this way, we maintain a consistent formal logic: the channel is the collection of all the discrete interfaces that can be individually or collectively controlled.

In practice, this model accounts for the selective nature of modern cyber-attacks, where the data flow for specific publishers or subscribers may be compromised while other participants remain entirely untouched. By moving away from the all-or-nothing approach of traditional network security models, we can more accurately describe decoupled architectures where the individual interfaces are the primary means of communication.

6.2 Refined architectural model

In the light of the pub/sub Dolev-Yao model, we must evolve our formal representation of system architecture to accommodate the complexities of modern networks. In Chapter 4 we defined interface systems as directed graphs, where nodes represent system actors (e.g. servers, sensors, or human users) and edges represent the discrete interfaces between them. In that one-to-one model, a directed arc $A \rightarrow B$ signifies a unique, coupled data flow from actor A to actor B .

While sufficient for point-to-point protocols, this graph-based approach fails to capture the “one-to-many” or “many-to-many” dynamics inherent to pub/sub systems. To address this, we move from the domain of simple graphs to the domain of *directed hypergraphs*. In our extended model, we maintain the set of nodes V representing system actors, but we redefine the connectivity using *directed hyperarcs*.

A hyperarc is an extension of an edge that can connect an arbitrary set of source nodes to an arbitrary set of destination nodes (Figure 6.2). We formally define a *topic interface* as a directed hyperarc $E = (T, H)$, where:

- the *tail set* (T) represents the set of actors authorised to publish on a specific topic;
- the *head set* (H) represents the set of actors subscribed to that topic, receiving any data published by any member of T .

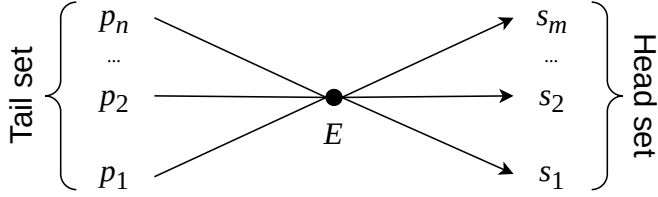


Figure 6.2: A generic hyperarc E . Its tail set is $T = \{p_1, p_2, \dots, p_n\}$; its head set is $H = \{s_1, s_2, \dots, s_m\}$.

This shift allows us to represent the spatial decoupling of the pub/sub paradigm. A publisher in T does not need a direct edge to every subscriber in H : the hyperarc itself acts as the broker that facilitates the distribution.

By using hypergraphs, we can elegantly express different fundamental communication patterns that are ubiquitous in IoT and IIoT environments:

Broadcast with a hyperarc where $T = \{A\}$ and $H = \{V \setminus A\}$. This represents for example a Wi-Fi Access Point broadcasting an SSID or a beacon;

Multicast with a hyperarc where H is a specific subset of V . For example an industrial controller sending a “Stop” command to a specific cluster of actuators;

Many-to-One with a hyperarc where T is a set of sensors and $H = \{Gateway\}$. This can represent multiple temperature sensors publishing telemetry to a single monitoring tool;

Unicast with a hyperarc with exactly one node in its tail set ($|T| = 1$) and exactly one node in its head set ($|H| = 1$), thus degenerating into a standard directed arc.

The shift to hyperarcs fundamentally alters how we define the concept of “interface compromise.” In a standard graph, an attacker capturing an arc only affects the (coupled) communication between two specific nodes. In our hypergraph model, the stakes are significantly higher: compromising a publisher or a subscriber may gain an attacker access to a much wider set of actors, since the communication is decoupled.

As established previously, we extend the model rather than replace it. A complete system is now represented as a hypergraph, containing both standard directed arcs (degenerate hyperarcs) for coupled, point-to-point interactions (e.g. a user connecting via a remote shell to a specific robot) and directed hyperarcs for uncoupled, topic-based interactions (e.g. a robot publishing its coordinates to a global choreography topic).

This hybrid approach is essential for identifying transitive vulnerabilities. For example, an attacker might use a broadcast hyperarc to spoof sensor data, which then causes a controller to initiate a vulnerable point-to-point service call to an actuator. Without the extension, this attack path would be invisible to formal verification tools.

6.2.1 Differences with established models

Historically, in the field of software architecture modelling via graph grammars, the literature is divided into two methodologies, each offering a distinct view upon system structure and connectivity. To fully appreciate our proposed hypergraph model, it is essential to contextualise it within these traditional frameworks: Le Métayer (1998) and Hirsch et al. (1998).

The first and most widespread approach, presented in Le Métayer (1998), utilises the classic definition of a graph $G = (V, E)$. Here, the nodes (V) represent the functional components of the system (e.g. a client or a microservice), while the edges (E) represent the communication links or dependencies between them.

This methodology is simple and intuitive: it mirrors the way engineers naturally sketch system topologies, documenting the architectural elements and their direct interconnections. For instance, a basic web architecture, consisting of a client, a load balancer, and a server, is rendered as three distinct nodes connected by arcs representing HTTP requests. While effective for high-level mapping, this approach often struggles to formally define the internal structure of the components themselves or the specific “ports” through which they communicate (Figure 6.3a).

The second approach, from Hirsch et al. (1998), leverages the mathematical power of hypergraphs, defined as (V, E, att) . In this paradigm,

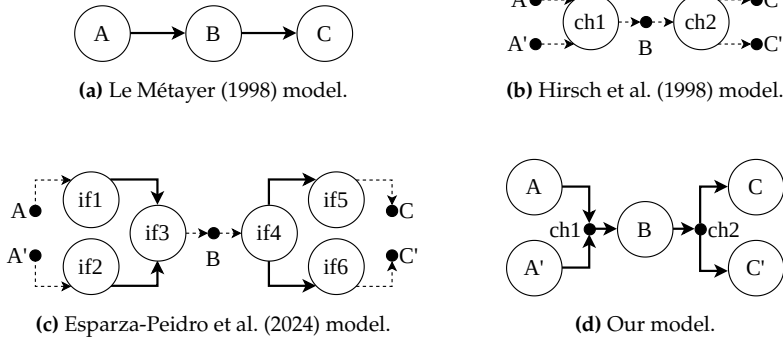


Figure 6.3: Summary of different interface models in literature.

the roles are reversed: components are represented by hyperedges (E), and the communication ports or hubs are represented by nodes (V). It is visually represented in Figure 6.3b. An hyperedge is viewed as an atomic entity with an ordered set of “tentacles.” These tentacles, dashed in the figure, attach to nodes via an attachment mapping $att : E \rightarrow V^*$. If two hyperedges are attached to the same node a communication link is established: in this view, the nodes act as connection hubs or synchronisation points. This method is often considered more expressive than the former because it allows the topology and the external interface of the components to be defined within a single, unified formal framework.

Coming to more recent work, Esparza-Peidro et al. (2024) (Figure 6.3c) tries to push the boundaries of the model by Hirsch et al. to address modern dynamic systems. This approach adopts a slightly different technique to model interconnections: while traditionally each hyperedge is assigned a specific number of source and destination attachment nodes to formally model the endpoints of the component, they connect components using additional directed arcs. These arcs explicitly join an “out” endpoint of one component to an “in” endpoint of another. Visually, dashed arrows represent the tentacles of the hyperedge (i.e. the binding of a component to its ports), while thick arrows represent regular edges (i.e. the network link between components).

By modelling both the internal endpoint and the external connector, the Esparza-Peidro et al. (2024) model gains the ability to analyse different types of failures: internal component failures (a hyperedge internal logic error), interface failures (a failure of the attachment node) or channel failures (a failure of the explicit arc connecting two nodes).

Our model. Contrary to what is done in Esparza-Peidro et al. (2024), we propose an extension of the Le Métayer (1998) model, which can still be viewed as the dual to the Esparza-Peidro et al. approach.

In our model, presented in Figure 6.3d, we maintain the intuitive mapping where nodes (V) represent system actors or components. However, we enhance the communication layer by replacing simple directed edges with directed hyperarcs. This creates a more versatile formal system where the complexity of the interaction is defined by the structure of the arc, rather than the internal tentacles of the component.

By using this approach, we can represent the entire spectrum of IIoT communications within a single hypergraph $H = (V, E)$, where the nature of the communication is determined by the cardinality of the hyperarc's head and tail sets. Point-to-point (unicast) communications are represented by degenerate hyperarcs: allowing us to maintain compatibility with classical coupled models where specific actors (like a client and a server) exchange direct messages. Hyperarcs with more than one head or tail represent decoupled interfaces, with enough flexibility to encompass various non-unicast communication patterns, including topic-based publish/subscribe channels, multicast groups, and broadcast mediums.

Our preference for the node-as-component view over the hyperedge-as-component view is driven by the fact that representing communication via (hyper)arcs perfectly aligns with the pub/sub Dolev-Yao model.

Moreover, from an automatic verification point of view, since ProVerif is based on the applied π -calculus, which treats processes as the primary entities that execute logic and channels as the medium for communication, we can map each node directly to a ProVerif process and each hyperarc as a set of channels. In the Esparza-Peidro et al. (2024) model the mapping would be more convoluted as processes must also act as links.

Also, ProVerif’s Dolev-Yao adversary automatically gains knowledge of any message sent over public channels. In a hyperedge-as-component model, defining what an attacker sees when they intercept a port node can be more complex.

Finally, by representing the interface as a communication link rather than a node, we preserve the semantics established in Chapter 4. In our framework, nodes remain exclusively the system actors, i.e. the entities that possess state, execute logic, and hold cryptographic secrets. The arcs, whether they are simple one-to-one links or more complex hyperarcs, represent the data flow between those actors. This consistency is important for maintaining a unified formal theory, allowing us to shift our analysis from isolated point-to-point interactions to distributed IoT ecosystems without fundamentally redefining what a component is.

6.3 Interface compromise

In order to maintain the rigour of our formal analysis, we must also extend the assumptions regarding interface compromise when transitioning from simple arcs to hyperarcs. This is necessary because interactions within our hypergraphs are inherently more complex and are logically mediated by a shared topic rather than a direct link. To analyse this, we can conceptualise a directed hyperarc as being composed of N distinct interfaces, where N corresponds to the combined cardinality of the head and tail sets. These interfaces are divided into two specific types: output interfaces, which represent the tail set of actors who possess the authority to publish data, and input interfaces, which constitute the head set of actors who subscribe to the topic and receive that data.

The semantic implications of an adversary’s actions shift depending on which of these levels is compromised. Unlike point-to-point communication, where a single breach affects a single stream, the topic acts as a functional mediator that manages data distribution. In a secure environment, the topic ensures a *copy-and-distribute* property, where every legitimate subscriber in the head set receives an identical and untampered version of the message published by the tail set.

Suppose an attacker manages to gain read access to the output interface of a component, i.e. an element in the tail set of a hyperarc. In this scenario, the adversary achieves a position of upstream observation granting them read-only visibility over all messages published by that *specific* interface as they are disseminated to every connected subscriber in the head set. This represents a targeted eavesdropping capability where the attacker monitors the data production of a single source and is not too different from the usual interface compromise for coupled systems.

Conversely, if we consider a scenario where the attacker gains read access to an input interface, the security implications are significantly broader. Because the underlying topic mechanism copies data from every publisher to every subscriber, a read compromise at the input level ensures the adversary can intercept the totality of the communication flowing from all output interfaces to the input interface.

Furthermore, by the same logic, gaining read access to a single input interface effectively grants read access to all other input interfaces within the same topic. In essence, by controlling a single subscriber’s viewpoint, the attacker leverages the topic’s inherent copy-and-distribute semantics to monitor the entire broadcast domain.

More formally, we can state the following *read amplification* law: given a hyperedge $E = (T, H)$,

$$(\exists i \in H : \mathcal{A}_{\text{in:RO}}^i) \implies (\forall t \in T : \mathcal{A}_{\text{out:RO}}^t) \wedge (\forall h \in H : \mathcal{A}_{\text{in:RO}}^h) \quad (6.1)$$

This distinction is very important for formal verification: while a tail-set compromise yields information about a specific source, a head-set compromise exposes the entire shared medium. Within this framework, *the confidentiality of the head set is critical for the system*: controlling a single input interface with read access effectively allows an adversary to read all messages published to that topic, confirming that, in decoupled IoT environments, an attacker’s point of entry determines whether their view is localised or global.

The implications of write access within our hypergraph model are even more significant, as they expose the implicit fan-out power of decoupled communication: if an adversary gains write access to an output interface,

they effectively acquire a “force-multiplier” for their attack. Because any message published to the topic is automatically replicated across the entire network, controlling a single output interface grants the attacker the ability to inject data into an unbounded number of subscribers.

We can formally state this fact as *write amplification* law: given a hyper-edge $E = (T, H)$,

$$(\exists i \in T : \mathcal{A}_{\text{out:WO}}^i) \implies \forall h \in H : \mathcal{A}_{\text{in:WO}}^h \quad (6.2)$$

In contrast, gaining write-only control over an input interface yields almost no strategic advantage in a decoupled system. Since data flows from the topic to the subscribers and not the reverse, an attacker who can write to an input interface is essentially manipulating a single node as if it was in a traditional one-to-one communication: their injected messages do not propagate back through the topic to other components. This asymmetry highlights another fundamental security principle for topic-based architectures: when dealing with active attacks, *the integrity of the tail set is the foundation of the entire topic’s security*.

Within the framework of decoupled architectures, the traditional parity between input and output security is replaced by a distinct risk asymmetry. This manifests as a cross-dependency between secrecy and integrity, where the severity of a compromise is dependent on the interface’s role within the topic.

Thus, a read-compromise is more dangerous at the subscriber level (the input interface) because the topic’s mediation logic ensures that every subscriber receives a comprehensive stream of data from every authorised publisher, granting an eavesdropper a global view of all communication. Conversely, a write-compromise is far more severe at the publisher level (the output interface) because the adversary can exploit the topic’s fan-out mechanism to automatically propagate payloads to an unbounded number of subscribers, effectively weaponising the infrastructure.

Consequently, both levels of the hyperarc must be treated as critical security boundaries: while securing the publishers protects data integrity across the system, securing the subscribers ensures data confidentiality of the entire broadcast domain.

	Output interface	Input interface
Read compromise	Read access from single interface	<i>Read access from full tail and head sets (Law 6.1)</i>
Write compromise	<i>Write access to full head set (Law 6.2)</i>	Write access to single interface

Table 6.1: Implications at various levels of access compromise. Notice the asymmetry: gaining read access to an interface in the head set yields full read access to all data originating from interfaces in the tail set; gaining write access in the tail set allows attackers to write to all interfaces in the head set.

Table 6.1 summarises the security implications for each level of interface compromise, detailing how the adversary’s impact scales based on the direction of the data flow and their level of access.

It is important to note that the Laws 6.1 and 6.2 exert an influence on the verification of scenarios: specifically, a scenario is considered valid only if it is closed under the constraints defined by the laws themselves. Consequently, the search space is restricted because the laws impose a logical structure that filters out invalid or inconsistent states. By ensuring that the model adheres to these structural invariants, we can guarantee that the automated verification process focuses exclusively on architecturally sound configurations, thus enhancing the efficiency of the analysis.

Still, for standard one-to-one interfaces, we simplify the representation by modelling only a single interface point as in a traditional unicast link the communication is coupled, meaning that the output interface of the sender and the input interface of the receiver are logically inseparable in terms of security compromise. If an attacker gains access to this link, they effectively control both sides of the channel: compromising the sender’s egress point is functionally identical to compromising the receiver’s ingress point. Therefore, modelling multiple distinct interfaces for a single one-to-one link adds unnecessary complexity to the state space without providing additional security benefits: we keep our original interface definition for the degenerate hyperarcs.

6.4 Running example: ABCAST with logger

The Atomic Broadcast (ABCAST) protocol, as presented in Birman et al. (1987), is designed to provide high-availability and fault tolerance in distributed systems by ensuring that all operational processes in a group receive the same set of messages in the exact same order.

Structurally, ABCAST is built upon a two-phase agreement logic: when a message is first multicast to the group, it is initially marked as *undeliverable* and assigned a temporary sequence number by each recipient based on their local counters. The recipients then return these suggested timestamps to the sender, who selects the maximum value among all suggestions and multicasts this final sequence number back to the group. Only upon receiving this final commitment do the processes reorder the messages in their local queues and deliver them to the application. The pseudocode for ABCAST can be found in Figure 6.4.

This mechanism effectively masks the asynchronous nature of the distributed network, providing a linearizable view of events, even in the presence of process failures or network delays. It is a classic implementation of a virtual synchrony model, where the system behaves as if every member is moving through a sequence of consistent, discrete states.

The motivation for selecting a protocol involving ABCAST as our primary running example throughout this formal analysis originates from its structural elegance: unlike simpler protocols that rely exclusively on one mode of interaction, ABCAST provides a testing ground for our hypergraph-based framework because it utilises both multicast and unicast communications. The initial dissemination and final commitment phases are modelled naturally as “true” hyperarcs, representing the one-to-many interactions with the topic interface. In contrast, the acknowledgment phase, i.e. when individual members send their proposed timestamps back to the sender, is represented by point-to-point edges.

This hybrid nature allows us to demonstrate the versatility of our extended model, proving that we can verify the security and consistency of coupled one-to-one links and decoupled topic-based interfaces within the same formal environment. Furthermore, ABCAST is simple enough to

Procedure ABCAST(m): ▷ Sender's side
broadcast m to all participants
await participant acknowledgments with proposed priorities
 $p_{final} \leftarrow \max(\text{proposed priorities})$
broadcast p_{final} to all participants

Procedure RECEIVER(): ▷ Participant's side
await m from sender and **store** m in temporary buffer
 $p_{prop} \leftarrow \text{GETNEXTLOCALPRIORITY}()$
send p_{prop} to sender
await p_{final} from sender
mark m as deliverable with priority p_{final}
reorder buffer and **deliver** all deliverable messages in order

Figure 6.4: Pseudocode for the ABCAST protocol.

avoid state-space explosion in symbolic solvers like ProVerif, yet sophisticated enough to exhibit vulnerabilities we seek to analyse when interface compromises occur.

The architecture for our running example protocol is presented in Figure 6.5. The system's components are:

Sender It initiates the protocol: it sends a message m to all participants, it waits for all the priority proposals, it calculates the maximum $p = \max(p_1, p_2, \dots, p_n)$, and it broadcasts the final priority p to commit the message order;

Participant 1 It follows the standard protocol: receives m , sends a proposed priority p_1 , and waits for the final priority. We assume, without loss of generality, that its priority is selected by the sender;

Participant 2 It also follows the ABCAST protocol;

Participant 3 It follows the protocol. Moreover, it sends an unrelated message to the Logger;

Logger It is a simple “sink” process, it waits a message from a participant and writes it to a log file.

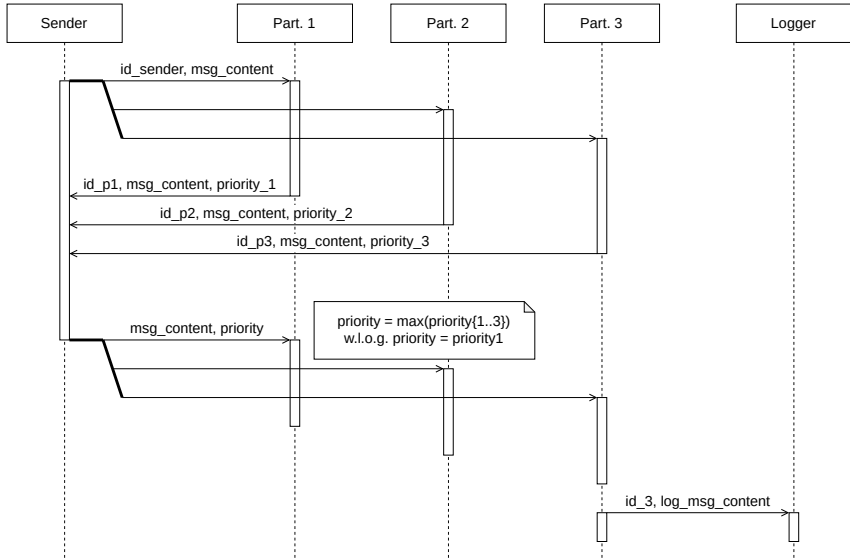


Figure 6.6: Sequence diagram of the ABCAST with logger protocol. Visually, the thick, multi-headed arrow represents a topic message.

linear sequence. Finally, the third participant subsequently transmits a log message to a dedicated Logger component via a point-to-point edge, which the Logger then receives and processes. This specific sequence is illustrated in the diagram provided in Figure 6.6.

6.5 ProVerif formalisation of ABCAST

In ProVerif, modelling a decoupled communication medium requires the introduction of a dedicated process to function as a broker or mediator. For each individual topic, we define a specialised *topic process* that explicitly manages the replication and dissemination of data from multiple senders to the corresponding set of receivers, while exposing these interactions to our security analysis framework.

The implementation of the topic process in our architecture (Figure 6.6) is detailed in Listing 6.1.

```

1 let topic_s_to_r() =
2   in(s_t, d:bitstring);
3   _OUT('S_T', (d))
4   ((
5     out(t_r1, d);
6     _IN('T_R1', (d))
7     0
8   ) | (
9     out(t_r2, d);
10    _IN('T_R2', (d))
11    0
12  ) | (
13    out(t_r3, d);
14    _IN('T_R3', (d))
15    0
16  ) ).

```

Listing 6.1: ProVerif code for the topic process for our architecture.

Unlike standard point-to-point channels in ProVerif which are implicit, this model treats the topic as an active entity (`topic_s_to_r`). It listens on a specific input channel `s_t` (representing the sender to topic edge) for a generic bitstring. Using the bitstring type is important here: it ensures the topic is type-agnostic and thus capable of carrying any structured message without needing a different process for every data type.

Once a message is received, the process performs multiple out operations. This effectively models the one-to-many fan-out property of the hyperarc. By sending the exact same bitstring to `t_r1`, `t_r2`, and `t_r3` (the three topic to participant edges), the model enforces data consistency, guaranteeing that all subscribers receive an identical copy of the message.

The use of the `m4` macros `_IN` and `_OUT` is the most important part of the security verification process. These are the instrumented hooks (identical to what is used for the CIE verification) used to model compromise:

`_OUT('T_Rn', (d))` signals that a message has entered the topic via the output interface. In ProVerif, we can map this to a leak to the public channel, if the attacker is able to read the output interface;

`_IN('S_T', (d))` represents the output interfaces: if the attacker has read access to the interface, this function ensures the adversary gains

knowledge of the bitstring term. By placing these hooks after each output, we can selectively model an attacker who has compromised a specific subscriber's link.

By reading and writing bitstrings, we align the topic with the Dolev-Yao adversary model. Since everything is a bitstring, ProVerif's attacker can attempt to: intercept the bitstring if the channel is public, inject its own bitstring if it gains output capabilities, or deconstruct the bitstring if it knows the underlying constructors (like encryption or signing keys).

This setup allows us to prove properties like *topic integrity*: for example, we can check if it is possible for an attacker to cause `_OUT('T_R1', d1)` and `_OUT('T_R2', d2)` to trigger with different values ($d1 \neq d2$) from the same initial input.

6.6 Verification

In complex systems, the security stance is rarely defined by a single invariant: rather, it relies on a composite of multiple properties such as secrecy, integrity, and authenticity. Verifying these properties in isolation can be misleading because security is usually non-compositional: a system is not “partially secure” if it loses one security property while maintaining others. In practice, we are often indifferent to which specific property is violated, as the breach of any single invariant means the adversary is able to bypass at least some protections. Still, this assumption does not limit the model's power: if specific insights are required, we retain the ability to verify individual properties in isolation.

To address this, we have extended our verification mechanism to support simultaneous multi-query assessment: the protocol is evaluated against the entire set of properties at once. The verification logic follows a strict-safety principle: if an attack is identified for even a single query, the entire protocol is immediately flagged as prone to attacks (i.e. the one identified by the specific query). Conversely, the system returns the “Safe” or “Cannot be proven” results only if the protocol is verified so for every individual query across all possible execution paths.

This approach is further justified by the fact that executing multiple queries within a single verification pass often significantly optimises the underlying computation. Symbolic provers are designed to share and reuse intermediate results, such as the derivation of internal Horn clauses across different queries in ProVerif. For instance, by analysing properties like secrecy and integrity simultaneously, the engine can leverage the overlap in their proof paths: e.g. the fact that a specific message is unreachable to the attacker may serve as a premise for proving multiple distinct security invariants. This cross-query optimisation accelerates the verification time and also often yields a more cohesive and computationally efficient security proof, as the tool avoids the effort of re-calculating the same protocol behaviours from scratch for every individual property.

To evaluate the security and correctness of the ABCAST-based protocol as modelled in Figure 6.6, we define a set of correspondence assertions in ProVerif. These queries allow us to verify that the protocol preserves its fundamental distributed invariants even when subjected to potential interface compromises:

Total Order Agreement The core requirement of any atomic broadcast is that all participants agree on the order of message delivery: if two honest processes, P_1 and P_2 , deliver the same message m , they must have assigned it the exact same priority $n_1 = n_2$ at the end of the protocol. This ensures that the two-phase priority negotiation successfully prevents state divergence across the topic’s subscribers:

$$\begin{aligned} \text{query } & p1 : \text{process_id}, p2 : \text{process_id}, m : \text{msg_content}, \\ & n1 : \text{priority}, n2 : \text{priority}; \\ & \text{event}(\text{Delivered}(p1, m, n1)) \wedge \\ & \text{event}(\text{Delivered}(p2, m, n2)) \Rightarrow n1 = n2. \end{aligned}$$

Legitimate Origin Another requirement is that the system is not susceptible to the injection of messages: we want to verify that every delivery is preceded by a legitimate (i.e. initiated by the Sender) broadcast. The query guarantees that if any process delivers a message m , that message must have originated from a legitimate send

event by an authorised publisher. It effectively tests whether a write-compromise at the publisher level would allow an adversary to force the delivery of fabricated data:

$$\begin{aligned} &\text{query } p1 : \text{process_id}, m : \text{msg_content}, n1 : \text{priority}; \\ &\quad \text{event}(\text{Delivered}(p1, m, n1)) \Rightarrow \text{event}(\text{Sent}(m)). \end{aligned}$$

Priority Validity The protocol logic dictates that the final sequence number must be derived from the proposals of the participants: we want to ensure that the final priority must be one of the participants' suggestions. The query verifies that an attacker cannot inject an arbitrary priority value to reorder messages or stall the system, confirming that the sender correctly selects the “max” priority from the set of valid local proposals;

$$\begin{aligned} &\text{query } p1 : \text{process_id}, m : \text{msg_content}, n1 : \text{priority}; \\ &\quad \text{event}(\text{Delivered}(p1, m, n1)) \Rightarrow \text{event}(\text{LocalPriority}(n1)). \end{aligned}$$

Logging Consistency The final requirement is to ensure the integrity of the secondary unicast communication channel used for logging. The query tries to confirm that if a log message is delivered, it was originally sent (similarly to the Legitimate Origin query). By checking the correspondence between the log delivery and sending, we ensure that the point-to-point link to the Logger component is secure against tampering.

$$\begin{aligned} &\text{query } p1 : \text{process_id}, m : \text{msg_content}, n1 : \text{priority}; \\ &\quad \text{event}(\text{LogDelivered}(p1, m)) \Rightarrow \text{event}(\text{LogSent}(m)). \end{aligned}$$

Given our interface model for the ABCAST protocol (Figure 6.5), which consists of eight unidirectional interfaces, the theoretical search space comprises $4^8 = 65\,536$ potential scenarios. This accounts for assigning one of four distinct access levels (none, RO, WO, RW), to each of the eight information flows. However, this upper bound can be significantly reduced by excluding write-only configurations, following the same logic applied to the CIE case (Section 4.4). Consequently, the automated generation yields a refined set of $3^8 = 6\,561$ scenarios for formal verification.

Threat Scenario								ABCAST
						$\mathcal{A}_{out:RW}^{r3,l}$		✗
				$\mathcal{A}_{out:RW}^{r1,s}$				✗
			$\mathcal{A}_{in:RW}^{t,r3}$					✗
		$\mathcal{A}_{in:RW}^{t,r2}$						✗
	$\mathcal{A}_{in:RW}^{t,r1}$							✗
$\mathcal{A}_{out:RO}^{s,t}$	$\mathcal{A}_{in:RO}^{t,r1}$	$\mathcal{A}_{in:RO}^{t,r2}$	$\mathcal{A}_{in:RO}^{t,r3}$	$\mathcal{A}_{out:RO}^{r1,s}$	$\mathcal{A}_{out:RW}^{r2,s}$	$\mathcal{A}_{out:RW}^{r3,s}$	$\mathcal{A}_{out:RO}^{r3,l}$	✓
$\mathcal{A}_{out:RW}^{s,t}$								✗

Table 6.2: Automatic verification results for the ABCAST protocol. See Table 5.1 for the legend.

6.7 Results

The verification results for our ABCAST protocol are shown in Table 6.2.

The most straightforward failure occurs in the $\{\mathcal{A}_{out:RW}^{r3,l}\}$ scenario, that is the scenario where the attacker directly gains write access to the Logger’s input interface. As expected, the attacker can trivially forge log entries, directly violating the Logging Consistency query. This confirms that point-to-point unicast links in our model behave as traditional coupled interfaces: a localised compromise at the receiver’s end grants the attacker full control over the data arriving at that specific node.

The scenario in which the attacker controls the interface between the first Participant and the Sender $\{\mathcal{A}_{out:RW}^{r1,s}\}$ presents a more interesting attack on the priority negotiation phase. While the Log Consistency invariant remains intact, the Priority Validity query fails. In this scenario, the attacker intercepts a legitimate priority suggestion from a participant and replaces it with a fabricated value higher than any other proposed timestamp. Because the protocol (under our assumption) treats the sender as trusting the priority selection from the first Participant, the adversary effectively hijacks the priority: by inflating it at the source, the attacker forces the entire group to synchronise on an illegitimate sequence number. The high level attack trace is visualised in Figure 6.7.

Another interesting failure occurs in the $\{\mathcal{A}_{in:RW}^{t,r2}\}$ scenario (and its symmetrical counterparts), which is the one where the attacker completely

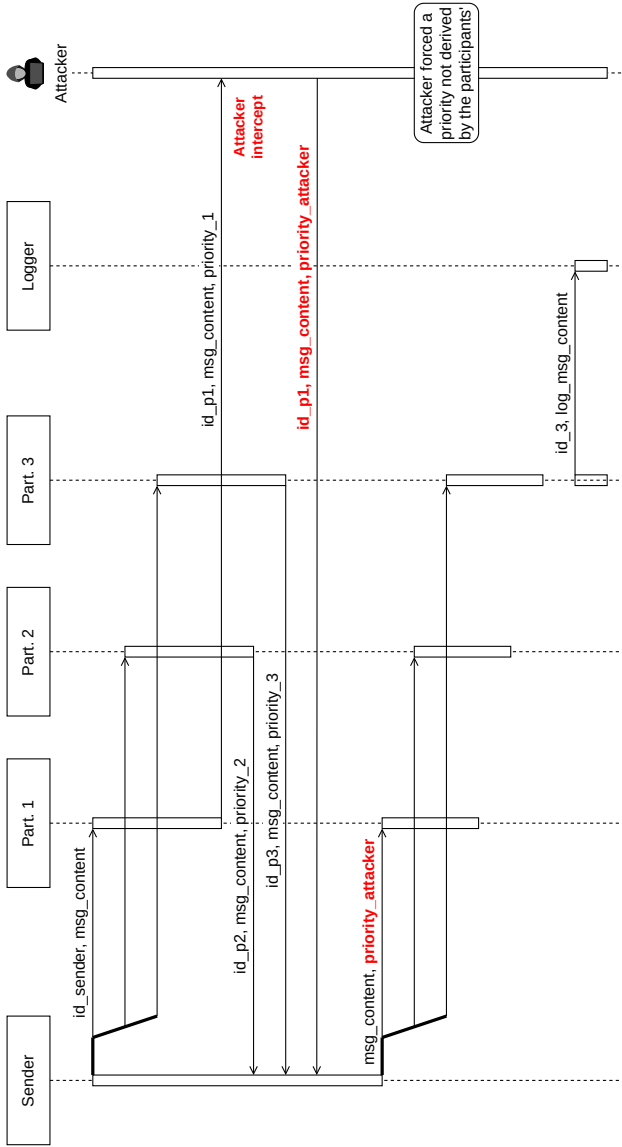


Figure 6.7: Overview of the ABCAST protocol attack in the $\{\mathcal{A}_{\text{out:RW}}^{r1_s}\}$ scenario.

controls an input interface from the topic to a Participant. The identified attack violates the Total Order Agreement property through a split-view attack and its trace is shown in Figure 6.8.

Firstly, the attacker intercepts a legitimate message from the sender on the compromised channel leading to the Participant. This is done to acquire knowledge of the message content, in order to masquerade as the Sender and force the Participant to accept a malicious, attacker-defined priority. Meanwhile, other Participants receive the authentic commit message from the topic with the correct priority.

This results in a state divergence where honest nodes deliver the same message but assign it different sequence numbers. This case perfectly illustrates the read/write asymmetry of the hyperarc: while a write-compromise at the publisher level (i.e. the $\{\mathcal{A}_{in:RW}^{s,t}\}$ scenario, where the attacker effectively controls the sender's output) affects everyone, a write-compromise at the subscriber level allows an attacker to isolate and deceive specific nodes, breaking the atomic guarantee of the ABCAST.

Possible solutions. We can determine the most effective remediation strategy to the vulnerabilities above identified. Following our methodology (Figure 1.2), we can intervene at two distinct levels:

- we can harden the protocol by incorporating additional security metadata, such as digital signatures, into its exchanges. However, this approach requires modifying the protocol's semantics and necessitates a complete refactor of its implementation;
- alternatively, we can introduce an intermediary layer adjacent to the protocol actors. This layer intercepts and modifies traffic to enforce security properties without altering the underlying protocol. While the protocol remains untouched, this adds new components to the overall system architecture.

In this case we prefer to intervene at the architectural level, in order not to modify the original protocol. We can still implement and verify the approach using ProVerif, but this requires the introduction of the concept of *service mesh*, which we will explore in the next chapter.

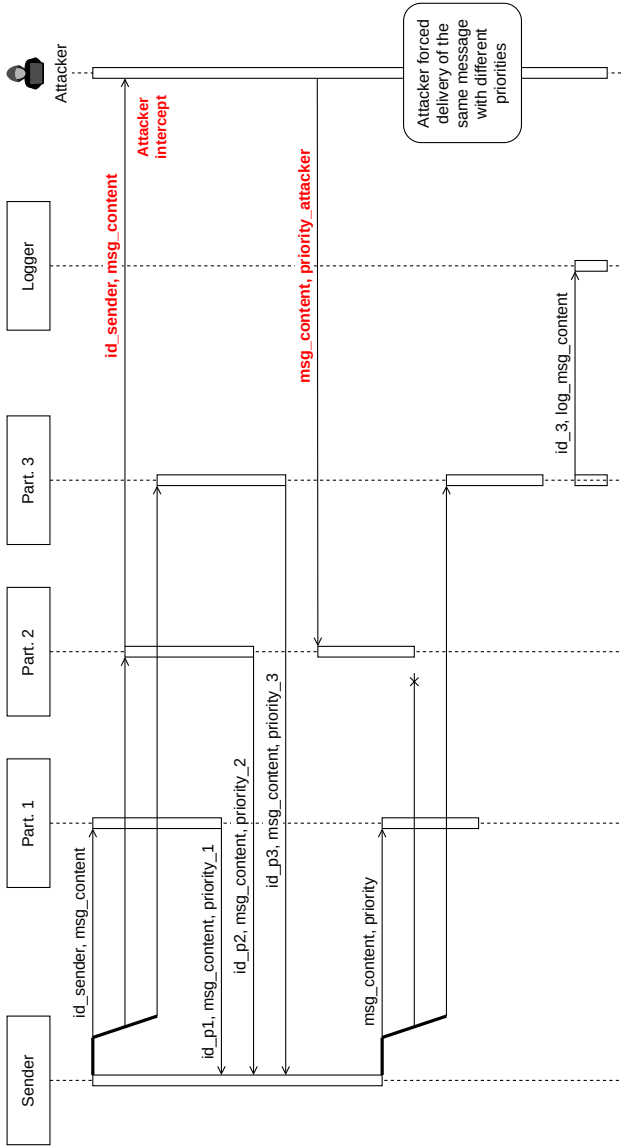


Figure 6.8: Overview of the ABCAST protocol attack in the $\{\mathcal{A}_{in:RW}^{1,2}\}$ scenario.

Chapter 7

Improving system architectures with service meshes

As distributed architectures have shifted from monolithic designs to fine-grained microservices, the network has become the backplane of application logic. This transition, however, introduced a significant security challenge: how to maintain a hardened perimeter when the inside of the application consists of potentially hundreds of ephemeral services communicating over an inherently untrusted network.

The *Service Mesh* (SM), introduced by platforms like Istio, Linkerd, and Consul, emerged both as a traffic management tool and as an important infrastructure layer for enforcing systemic security properties without requiring modifications to the application source code (Ganguli et al. 2021; Sheikh et al. 2018).

7.1 Service meshes

In microservice architectures, monolithic applications are decomposed into smaller and independent services, typically deployed via virtual machines or *containers*.

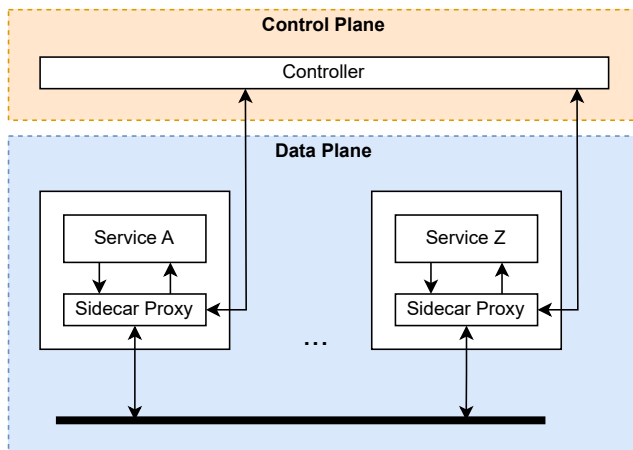


Figure 7.1: A generic service mesh representation.

While this paradigm enhances scalability, resilience, and development velocity, it introduces significant complexity in managing inter-service communication and security. Hardcoding these functionalities into each service implementation leads to redundancy, complicates maintenance, and increases the risk of security misconfigurations (W. Li et al. 2019).

To address these challenges, a service mesh decouples infrastructure logic from business logic. As shown in Figure 7.1, the architecture typically consists of two main components: a *controller* and a set of *sidecar proxies*. These proxies form a network (the *data plane*) that handles all inter-service request flows. By intercepting all incoming and outgoing traffic, each sidecar proxy enforces security and communication policies received from the *control plane*, ensuring the underlying services remain agnostic to these security requirements.

While SMs serve various purposes, our focus is on their role in *authentication* and *authorisation*. *Authentication* ensures that only legitimate, verified services can establish communication, while *authorisation* governs which specific actions a service or user is permitted to perform on a given resource. SMs provide significantly more fine-grained access control and identity management than traditional orchestrator-level network policies.

For instance, rather than requiring every microservice to manage its own cryptographic libraries and private keys, a SM can automatically sign outgoing messages. This ensures that even if a message is intercepted or routed through an untrusted intermediary, its provenance and integrity remain verifiable.

Another similar use case for the use of service meshes is the automated enforcement of mTLS. The mesh handles certificate issuance, rotation, and revocation, ensuring that every hop between services is encrypted and mutually authenticated by default. Additionally, service meshes can employ identity-based access control: instead of relying on fragile IP-based firewalls, service meshes can use stronger identities. This allows security architects to define access properties based on the identity of the service (e.g. “the billing service can only be called by the order processing service”), providing a zero-trust posture within the system.

Since the service mesh is, in fact, the network layer of the distributed system, it can provide mechanisms to enforce complex access control policies and observability across the entire hypergraph of services. By centralising the control plane, administrators can define global invariants, such as ensuring that sensitive user data never leaves a specific geographic boundary or that a service can only communicate with another one only using a specific API method.

Because the mesh acts as a mediator, it provides a consistent source of truth for security audits. Adding this pattern to our systems, every interaction, whether a multicast broadcast to a topic or a point-to-point unicast call, can be logged and analysed, potentially decorating it with additional ProVerif events or changing its structure.

Moreover, in the context of our research, the service mesh provides the perfect real-world justification for the node-as-component and hyperarc-as-interface model. In a service mesh, the node is the application container, and the interface is the proxy-managed connection. By formalising these interactions, we can prove that even if an attacker gains write access to a specific service interface, the mesh’s policy engine can prevent that compromise from fanning out across the network, thereby preserving the integrity of the global system.

7.2 Stateful access policies

While service meshes offer powerful capabilities, they require robust governance mechanisms to ensure consistency, security, and compliance across the system.

To ease these processes, *policy-based* approaches have gained significant traction (OpenFGA 2024; OpenPolicyAgent 2024; Pallewatta et al. 2024; Pang et al. 2019). This paradigm uses declarative, domain-specific languages to define the rules and constraints governing service behaviour: by decoupling policy definition and enforcement from implementation, organisations can centralise management, enforce uniform security standards, and minimise the risk of configuration errors.

A fundamental distinction exists between basic (stateless) access policies and *stateful access policies*. While basic policies provide the essential foundation of “who can talk to whom,” stateful policies introduce the dimension of “when and under what conditions.” By integrating these into the service mesh layer, we can move from simple identity verification to complex, behaviour-aware protocol enforcement.

Basic access policies are typically invariant and identity-centric. They typically function as a distributed firewall where the decision to allow a message is based solely on static attributes: the identity of the sender, the target interface, and perhaps the message type. These are often implemented via Access Control Lists (ACLs) or simple Role-Based Access Control (RBAC). The main limitation of a basic policy is that it cannot distinguish between a first-time request and a thousandth-time one. If a Participant is authorised to send a priority proposal, a basic policy will allow that Participant to send that proposal indefinitely, even if the protocol logic dictates that only one proposal should be sent per round.

Stateful policies, by contrast, maintain an internal memory of the protocol’s history. They allow the service mesh to enforce constraints based on the sequence and frequency of interactions. Consider a scenario involving an API rate-limiting policy, where a service is permitted only 100 interactions per day, with any subsequent requests blocked until a plan upgrade is performed.

In a traditional architecture lacking stateful infrastructure, implementing such a policy would necessitate modifications to the application source code. Developers would be forced to manually integrate counter logic, database lookups for usage tracking, and conditional branching to handle the quota exceeded errors within the business logic itself. This not only complicates the codebase but also tightly couples security and billing constraints with the application’s core functionality.

In contrast, a stateful service mesh proxy handles this logic entirely within the infrastructure layer. The application remains unaware of the limit, continuing to issue requests as normal. The proxy intercepts each outbound call, checks a persistent state store (the counter), and simply stops forwarding requests once the 100th interaction is recorded. By offloading this state management to the mesh, we ensure that policy enforcement is consistent across all services, regardless of the language or framework they are written in, and can be updated globally without code modification in the application.

Our framework extends beyond static identity-based security to support stateful access policies, which are essential for enforcing dynamic constraints during the protocol lifecycle. In our running ABCAST example we can introduce the enforcement of a “communication quarantine” to mitigate pre-commitment interference. Suppose we do not fully trust the participants and wish to enforce the following strict invariant: a Participant must only gain the ability to communicate with the Logger once the final priority has been committed by the Sender.

To implement this, we can leverage a service mesh as a distributed state monitor. This configuration would utilise two distinct service mesh definitions: one for the Sender and one for the Participants, coordinated through a shared state variable (a cell).

The enforcement mechanism operates through a synchronised state transition, defined by two moments:

Trigger on the outbound Sender service mesh. When the Sender issues the final commit message, its service mesh macro performs an atomic write to a state cell;

```

1  m4_define(`_IN_SMid', `in($1,$2'))
2  m4_define(`_OUT_SMid', `out($1,$2'))
3
4  m4_define(`_IN_SMsign', `in($1, ($2, signature:bitstring)); if
   checksign(signature, pk(k)) = h($3) then ');
5  m4_define(`_OUT_SMsign', `out($1, ($2, sign(h($2),k)));')

```

Listing 7.1: Different service meshes implementations.

Guard on the outbound Participant service mesh. Outbound message sends are redefined to check the state cell before allowing any communication. If the cell indicates that the commit has occurred, the macro intentionally fails to execute, effectively killing the process and thus its ability to send further messages.

This stateful approach is particularly effective in preventing byzantine behaviour: even if a participant’s application logic is compromised and attempts to send malicious data or spurious priority updates before the agreement phase, the service mesh layer acts as a gatekeeper and prevents unwanted communication.

7.3 Service mesh implementation in ProVerif

To implement the service mesh layer in ProVerif without manually rewriting every process, we utilise the m4 macro processor as a pre-processing engine. This allows us to abstract the communication primitives into programmable hooks, effectively decoupling the application logic from the security infrastructure.

By replacing the native ProVerif in and out primitives with the specific service mesh macros `_IN_SM` and `_OUT_SM`, we create an interception point for every message exchange. The macros in Listing 7.1 define two different service meshes (one for each pair).

The first service mesh, namely `SMid`, acts as a transparent proxy. It accepts two arguments, the channel name and the data variable, and simply executes the standard corresponding π -calculus operation. However, the power of this approach lies in its flexibility: because the application

processes use these macros instead of raw keywords, we can change the entire posture of the system by modifying only these two lines of code.

The second service mesh (`SMsign`) shows just that: it implements an automated security layer that enforces data integrity. The output macro encapsulates the outgoing data by appending a cryptographic signature, generated by signing a hash of the message (`h ($2)`) with a private key `k`. On the receiving end, the input macro intercepts the incoming bitstring, extracts the message and the signature, and utilises an if-then conditional to verify the signature against the expected key `pk (k)`. If the signature is invalid or the hash does not match, the process effectively halts (it is replaced by the null process), preventing the application logic from consuming unauthenticated or tampered data. By using these macros to inject this logic directly into the in and out primitives of the system, we can transform a standard protocol into a digitally signed one.

It is important to notice that this architecture ensures that the application processes remain entirely decoupled from the security material: they operate without any direct knowledge of cryptographic secrets, such as signing keys. By offloading these responsibilities to the service mesh macros, we maintain a robust defence-in-depth posture where even byzantine processes, i.e. nodes that are faulty or intentionally malicious, cannot easily forge authentication tokens or leak keys they never possessed. This separation of concerns allows the formal model to prove that the system's global integrity and secrecy can be preserved even when individual component logic is compromised or behaves unpredictably.

While the macro-based approach provides a clean abstraction, its primary advantage in symbolic verification is the prevention of state-space explosion: if we were to implement the service mesh using the standard (and more naïve) modelling technique, where each proxy is a standalone ProVerif process running in parallel with the application, we may encounter significant computational overhead. In a standard sidecar architecture, every application node is paired with a proxy node. In ProVerif, this would mean that for every process P_i representing business logic, we would need to define a corresponding SM_i to handle communication: this would double the number of active processes in the model.

Since ProVerif evaluates the interleaving of all possible process actions to find attack traces, the complexity grows exponentially. Each additional process introduces new sets of internal channels, message passing steps, and non-deterministic choices that the solver must explore. By using m4 macros, we perform in-place expansion: the service mesh logic is compiled directly into the original process during pre-processing, maintaining the original process count and keeping the state space manageable.

Moreover, ProVerif often struggles to resolve the reachability of a state due to an excessive number of internal communication steps. In a dual-process model (i.e. process to proxy to network), every single application message requires two hops and two internal synchronisations before it even reaches the topic or another peer. Finally, these internal hops between the application and the proxy are usually uninteresting from a security perspective, yet ProVerif must model them as potential points of failure or interleaving.

In our framework, stateful policies can be modelled by introducing m4 macros that can interact with ProVerif’s tables. By moving the state logic into the service mesh macros, we prevent modifications to the verification code. We can write the protocol logic as if it were a simple sequence of steps, while the service mesh ensures that the execution strictly adheres to the temporal and quantitative bounds required by our invariants.

The m4 macros to implement the two ProVerif service meshes for the described communication quarantine policy (Section 7.2) are shown in Listing 7.2. Notice that the SM for the Participant may lock up only if the interface that is trying to communicate is `inth`.

7.4 ABCAST with service meshes

Integrating the previously discussed service mesh architecture into our ABCAST protocol provides a transparent defence against the integrity-based attacks identified in our analysis. As demonstrated in Section 6.7, the primary vulnerabilities of the protocol arise from an adversary’s ability to intercept, modify, and re-inject messages such as priority proposals or commit signals across compromised interfaces.

```

1  table state_unlock().
2
3  m4_define(`_IN_SMpart', `in($1,$2)')
4  m4_define(`_OUT_SMpart',
5      `m4_ifelse(`$1', `inth',
6          `get state_unlocked() in out($1,$2);',
7          `out($1,$2);'
8      )
9  ')
10
11 m4_define(`_IN_SMsend', `in($1,$2)')
12 m4_define(`_OUT_SMsend',
13     `m4_ifelse(`$3', `UNLOCK',
14         `insert state_unlock(); out($1,$2);',
15         `out($1,$2);'
16     )'
17 )

```

Listing 7.2: Stateful policy service mesh implementation.

By employing the signing service mesh logic defined in Listing 7.1, we can effectively neutralise these threats without altering the underlying application code of the ABCAST participants. The security of this enhanced model relies on the automated decoration of every message with a cryptographic signature at the egress point and its verification at the ingress point. Because the signing keys reside within the service mesh layer rather than in the application, we achieve separation of concerns: the mesh applies security transformation on the flowing data, while protocol components can execute the original protocol without modification.

In the $\{\mathcal{A}_{\text{out:RW}}^{r1,s}\}$ scenario, where the attacker previously inflated priority values (Figure 6.7), the service mesh would enhance security: they cannot modify the priority value as any change would invalidate the signature, causing the input check at the Sender’s interface to fail and abort the protocol. Similarly, for topic-based failures like the one in the $\{\mathcal{A}_{\text{in:RW}}^{t,r2}\}$ scenario (Figure 6.8), the attacker can no longer masquerade as the sender: since the adversary lacks the private key, they cannot forge a commit message with a malicious priority. The Participant whose interface is compromised will only accept messages that possess a valid signature, ensuring it remains synchronised with the rest of the group.

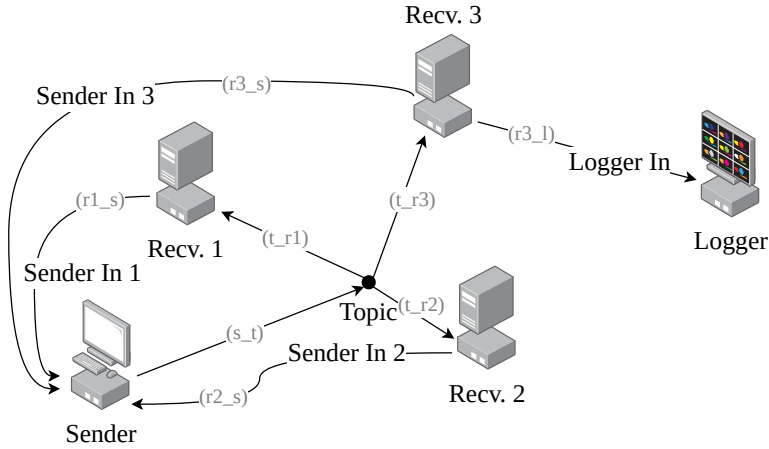


Figure 7.2: Interface model for ABCASTSM. Notice the data plane introduced by the service mesh sidecar proxies.

Another primary objective is to enforce an environment where messages are also subject to the communication quarantine stateful policy introduced in Section 7.2, which ensures that participants can communicate with the Logger only once the consensus phase is finalised.

The operational sequence of this enhanced ABCAST protocol, called ABCASTSM, remains identical to the standard version shown in Figure 6.6, but every network interaction is now wrapped in service mesh logic: every outbound message is automatically signed by the sender’s proxy using keys stored in the infrastructure layer; upon receipt, the destination’s service mesh verifies the signature before the application logic ever processes the data. Moreover, the SM enforces the quarantine invariant: unlike a stateless firewall, it tracks the protocol’s progression, and verifies that the protocol execution does not violate the constraints of the quarantine policy at any point. The interface model for ABCASTSM is shown in Figure 7.2; its sequence diagram in Figure 7.3.

We can verify the baseline example protocol (ABCAST) is compatible with the quarantine assertion by performing a sanity check in the empty (compromised-free) scenario before assessing the protocol against an

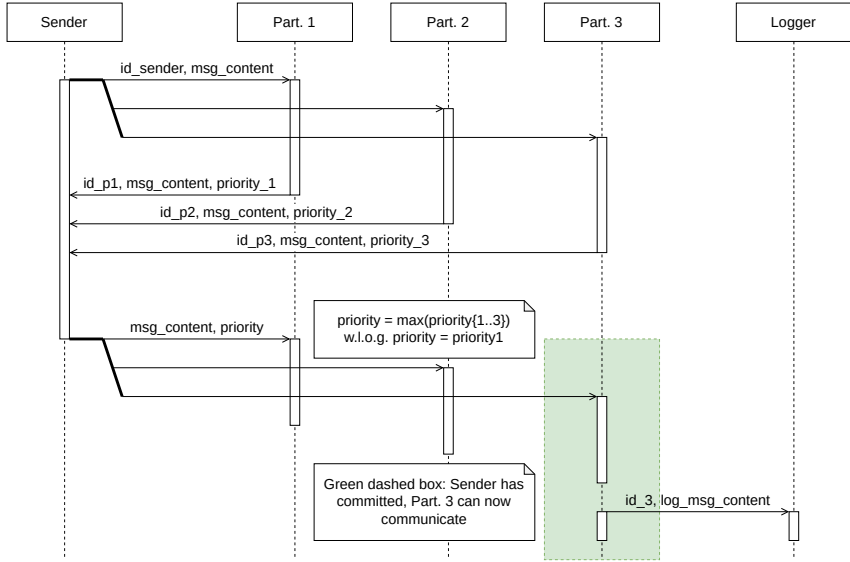


Figure 7.3: Sequence diagram of the ABCASTSM protocol. As in Figure 6.6, the thick arrow represents a topic message.

active adversary. This ensures that the protocol is functional under normal conditions, thus proving that the goals are reachable when no attacker is present. This sanity check aims to prove two simultaneous reachability properties: firstly, the protocol can successfully deliver the ABCAST message; secondly, the unicast logging operation completes successfully.

7.4.1 Results

As can be seen by the results in Table 7.1, this approach is able to fix the attacks present in the original protocol. These attacks originate from the ability to inject fabricated or forged messages that the adversary gains when compromising the logger input interface ($\{\mathcal{A}_{\text{out:RW}}^{r3_l}\}$), a participant-to-sender channel, (e.g. $\{\mathcal{A}_{\text{out:RW}}^{r1_s}\}$), or the input from the topic for a participant (e.g. $\{\mathcal{A}_{\text{in:RW}}^{t2}\}$).

This is because every outgoing message is now accompanied by its signature, generated by a key isolated within the service mesh, and thus

the adversary's ability to write to an interface is limited to replaying or dropping existing valid packets. Since they lack the private key, any attempt to modify the bitstring content results in a signature mismatch at the receiving sidecar proxy, which then drops the tampered data. Consequently, the ProVerif analysis now marks these previously vulnerable scenarios as safe, confirming that the protocol invariants are preserved even under some interface compromise.

While the service mesh provides a robust defence against localised (i.e. single interface) breaches, it does not, in fact, fix all the vulnerabilities: our analysis reveals that protocol failure can still occur when the adversary achieves multi-interface compromise.

This suggests that the service mesh effectively increases the effort needed by an adversary to break the protocol or induce an unsafe state by introducing the additional security layers that need to be bypassed.

A critical example of attacks that can arise with multi-interface compromise is found in the scenario $\{\mathcal{A}_{in:RO}^{t,r2}, \mathcal{A}_{out:RW}^{r3,1}\}$: in this case, the adversary can intercept messages from the topic output and exercise full control over the Logger's input interface. Notice that this scenario is symmetrical to all the others where an attack is found in ABCASTSM (Table 7.1).

Under these conditions, the attacker can execute a cross-interface replay attack. Because the Logger expects a couple of bitstrings representing a log entry, and the service mesh's commit message is also a signed couple of bitstrings, the attacker can intercept a validly signed commit message intended for a participant and re-inject it into the Logger's interface.

Since the message carries a valid signature from a trusted component (i.e. the sender's service mesh), the Logger's proxy verifies the signature and accepts the data as authentic. This causes the Logger to confuse the protocol logic, recording a commit value as if it were a log entry.

This demonstrates, unsurprisingly, that even with cryptographic signatures, if the service mesh or the protocol itself does not include context-specific tags (e.g. distinguishing a log type from a commit type) within the signed payload, an adversary can leverage a write-compromised interface to reuse legitimate messages from a read-compromised one. The attack is visualised in Figure 7.4.

Threat Scenario		ABCAST	ABCASTSM	ABCASTSM*
$\mathcal{A}_{out:RW}^{l1,s}$ $\mathcal{A}_{out:RW}^{l1,s}$ $\mathcal{A}_{out:RW}^{l2,s}$ $\mathcal{A}_{out:RW}^{l3,s}$	$\mathcal{A}_{out:RW}^{l3,I}$	$\overline{\mathbf{X}}$	$\checkmark$	$\checkmark$
	$\mathcal{A}_{out:RW}^{l3,I}$	$\overline{\mathbf{X}}$	$\checkmark$	$\checkmark$
	$\mathcal{A}_{out:RW}^{l3,I}$	$\mathbf{X}$	$\checkmark$	$\checkmark$
	$\mathcal{A}_{out:RW}^{l3,I}$	$\mathbf{X}$	$\overline{\mathbf{X}}$	$\checkmark$
$\mathcal{A}_{in:RO}^{l3}$ $\mathcal{A}_{in:RW}^{l3}$	$\mathcal{A}_{in:RW}^{l3,I}$	$\overline{\mathbf{X}}$	$\checkmark$	$\checkmark$
	$\mathcal{A}_{in:RW}^{l3,I}$	$\mathbf{X}$	$\overline{\mathbf{X}}$	$\checkmark$
	$\mathcal{A}_{in:RW}^{l3,I}$	$\mathbf{X}$	$\checkmark$	$\checkmark$
	$\mathcal{A}_{in:RW}^{l3,I}$	$\mathbf{X}$	$\overline{\mathbf{X}}$	$\checkmark$
$\mathcal{A}_{in:RO}^{l2}$ $\mathcal{A}_{in:RW}^{l2}$	$\mathcal{A}_{in:RW}^{l2,I}$	$\mathbf{X}$	$\overline{\mathbf{X}}$	$\checkmark$
	$\mathcal{A}_{in:RW}^{l2,I}$	$\mathbf{X}$	$\overline{\mathbf{X}}$	$\checkmark$
	$\mathcal{A}_{in:RW}^{l2,I}$	$\mathbf{X}$	$\checkmark$	$\checkmark$
	$\mathcal{A}_{in:RW}^{l2,I}$	$\mathbf{X}$	$\overline{\mathbf{X}}$	$\checkmark$
$\mathcal{A}_{out:RO}^{s,t}$ $\mathcal{A}_{out:RO}^{s,t}$ $\mathcal{A}_{out:RW}^{s,t}$ $\mathcal{A}_{out:RW}^{s,t}$	$\mathcal{A}_{out:RW}^{s,t}$	$\mathbf{X}$	$\overline{\mathbf{X}}$	$\checkmark$
	$\mathcal{A}_{out:RW}^{s,t}$	$\checkmark$	$\checkmark$	$\checkmark$
	$\mathcal{A}_{out:RW}^{s,t}$	$\overline{\mathbf{X}}$	$\checkmark$	$\checkmark$
	$\mathcal{A}_{out:RW}^{s,t}$	$\mathbf{X}$	$\checkmark$	$\checkmark$
$\mathcal{A}_{in:RW}^{s,t}$ $\mathcal{A}_{in:RW}^{s,t}$ $\mathcal{A}_{in:RW}^{s,t}$ $\mathcal{A}_{in:RW}^{s,t}$	$\mathcal{A}_{in:RW}^{s,t}$	$\mathbf{X}$	$\overline{\mathbf{X}}$	$\mathbf{X}$
	$\mathcal{A}_{in:RW}^{s,t}$	$\mathbf{X}$	$\overline{\mathbf{X}}$	$\mathbf{X}$
	$\mathcal{A}_{in:RW}^{s,t}$	$\mathbf{X}$	$\overline{\mathbf{X}}$	$\mathbf{X}$
	$\mathcal{A}_{in:RW}^{s,t}$	$\mathbf{X}$	$\overline{\mathbf{X}}$	$\mathbf{X}$

Table 7.1: Results comparison of the ABCAST protocols. See Table 5.1 for the legend.

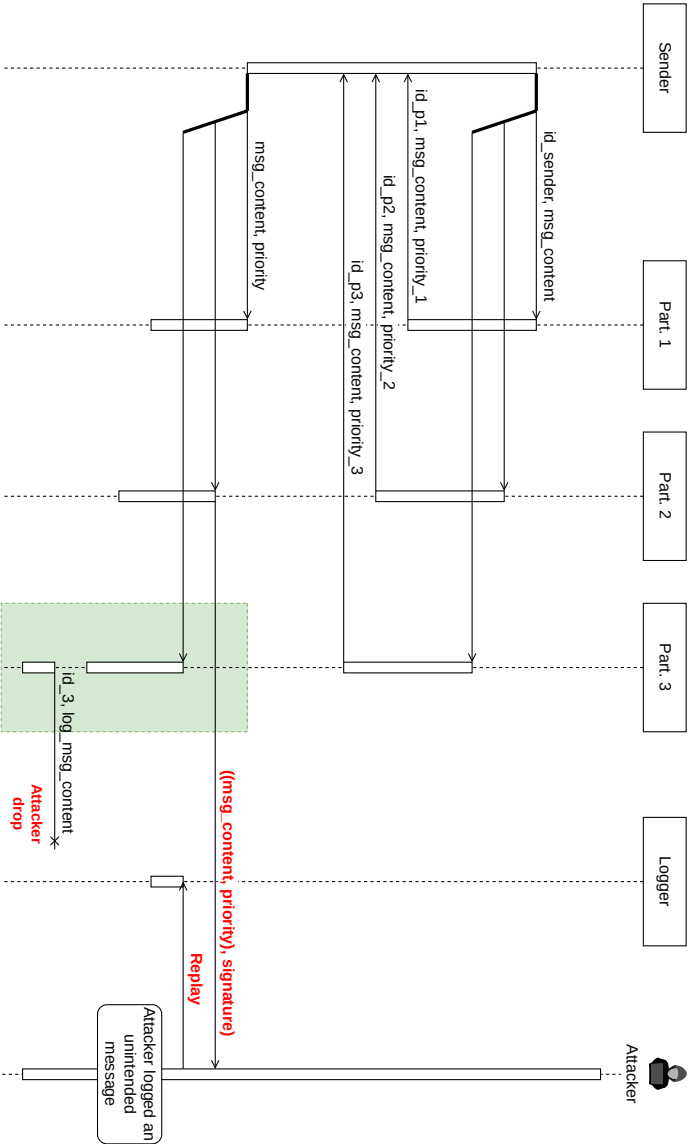


Figure 7.4: Overview of the ABCASTSM protocol attack in the $\{\mathcal{A}_{in:RO}^{I,R2}, \mathcal{A}_{out:RW}^{3,I}\}$ scenario.

7.4.2 ABCASTSM*, ABCASTSM improvement

In order to mitigate the cross-interface replay vulnerabilities identified in our previous analysis, we introduce message tags, for example at the protocol level, evolving our model into ABCASTSM*. This enhancement ensures that every message is cryptographically bound both to its content and to its specific intent and context.

In this model, the protocol is refined to include a unique constant representing the message type within the signed payload. By including these tags, we ensure that a message intercepted from a topic interface cannot be re-injected into the Logger interface, as the receiver will detect the mismatch and reject the message as invalid for that specific context.

As shown in Table 7.1, the ABCASTSM* protocol successfully withstands even stronger compromise scenarios, including the fully compromised case at read/write level. By combining signing, stateful policies, and message context separation via tags, we achieve a final protocol where the adversary can no longer forge, modify, or successfully replay messages across interfaces.

It is important to acknowledge that the perfect security results we achieve here are due to the ABCAST protocol's relative simplicity: because the protocol has a linear state progression and a small set of distinct message types, we can formally cover every edge case. In larger, more complex production environments, we would expect results more aligned with the improved CIE Level 3 results presented in Section 5.6. In such systems, while the use of service meshes may significantly harden the architecture against traditional network attacks, the vast number of interleaving processes and the possibility of complex logical race conditions mean that security is rarely absolute.

Chapter 8

Implementation of stateful policies for service meshes

A significant open problem in the literature regards the formalisation of stateful policies within service mesh architectures. Specifically, there is a lack of consensus on how to represent policies that depend on dynamic context, as well as how to use existing policy languages for expressing the transition and modification of state. To address these limitations, we propose a stateful policy engine designed to support the stateful service mesh configurations introduced in Chapter 7. This solution enables the enforcement of context-aware security properties, permitting the actual implementation of the improved protocols we analysed.

To develop our tool we employ Rego, that, among the various domain-specific languages that have been developed for policy authoring, has become a prominent choice within service mesh ecosystems (OpenPolicyAgent 2024) due to its expressive syntax and powerful evaluation engine that facilitate the implementation of complex policies designed to enforce diverse security, reliability, and performance constraints. However, Rego’s functional and declarative design, while capable of querying external data stores, is fundamentally restricted to read-only operations. This limitation poses a significant challenge when defining access policies that rely on mutable state (Baldo, Ion, Miculan, Paier, and Riccio 2025).

8.1 Security policies in distributed applications

In recent years, a significant focus of research and development has been dedicated to addressing the complexities of expressing security policies within distributed applications: current methodologies for managing these policies generally fall into two distinct categories, namely *Policy via Configuration Files* and *Policy-as-Code*.

Configuration files consist of structured sets of key-value pairs designed to dictate system behaviours and environmental parameters. This methodology is the standard for cloud-native infrastructures and service meshes: prominent platforms such as Istio (Sheikh et al. 2018) and Linkerd (Linkerd 2024) utilise these files to ensure that policies are encoded in a reproducible and structured format.

Despite their prevalence, configuration files are fundamentally static. They miss the native ability for conditional logic, complex data structures, or arithmetic operations. This lack of expressiveness makes them ill-suited for specifying the dynamic policy requirements often necessitated by modern distributed systems.

Conversely, the Policy-as-Code paradigm (Pallewatta et al. 2024) treats security policies as software, utilising specialised programming languages to define authorisation logic: this approach supports more sophisticated constructs such as conditional control flows and arithmetic operations, providing the expressiveness required for dynamic environments.

A foundational example of this paradigm is XACML (Lorch et al. 2003), which encodes access control policies in XML. XACML introduced the concept of obligations, actions the *Policy Enforcement Point* (PEP) must execute alongside the authorisation decision. However, the XACML specification remains abstract regarding the internal architecture of the authorisation agents, or *Policy Decision Points* (PDPs). To address the complexity of XACML, FACPL was introduced (Margheri et al. 2019), offering a more concise syntax supported by a rigorous denotational semantics for real-world policy modelling. More contemporary advancements include OpenFGA (OpenFGA 2024), a fine-grained engine inspired by Google’s Zanzibar (Pang et al. 2019) that focuses on relationship-based

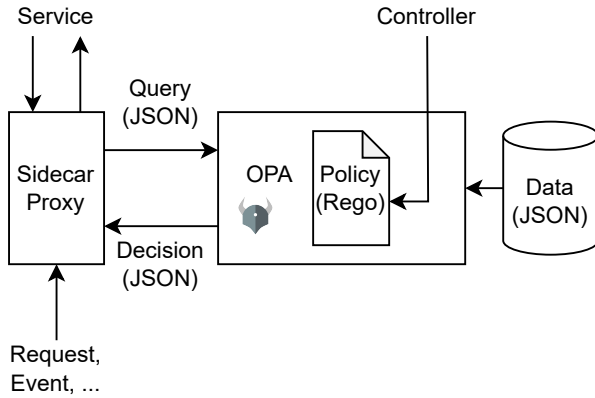


Figure 8.1: OPA's decision flow.

access control; Cedar (Cutler et al. 2024), developed by AWS Labs, a high-performance language for access control whose core semantics have been formally verified using the Lean proof assistant (Disselkoen et al. 2024).

Rego (OpenPolicyAgent 2024), part of the Open Policy Agent (OPA) project, has emerged as one of the most widely adopted languages in the Policy-as-Code ecosystem. Designed to manipulate semi-structured JSON data, Rego evaluates incoming requests and generates structured output that encapsulates the results of the policy logic.

This versatility has led to Rego's integration across diverse domains, including cloud compliance automation and authoritative nameserver architectures (Larisch et al. 2024; Oku et al. 2022; Paul et al. 2024). Its interoperability is further evidenced by the fact that policies can be bi-directionally translated between XACML and Rego.

Given this significant industry adoption and the increasing academic interest in the language, we focus our efforts on extending the capabilities of OPA, the official authorisation engine for Rego, to overcome its functional constraints.

Within the service mesh ecosystem, OPA is typically deployed as a microservice that sidecar proxies use to provide fine-grained authorisation logic (Figure 8.1).

A Rego policy consists of a set of *rules* that dictate the final outcome of an OPA evaluation (as illustrated in Listing 8.1). Each rule is structured with a head, defining the specific decision or value to be produced, and a body, which contains the conditions or queries that must be satisfied for the rule to trigger.

To resolve these decisions, OPA processes information from two primary sources: the incoming request (Listing 8.2) and any supplemental data (Listing 8.3) within the evaluation context.

While rules are commonly designed to return boolean values, Rego's functional nature allows for much greater flexibility: policies can return complex, structured results, such as specific permission sets or error messages, encapsulated within the final JSON output object (Listing 8.4).

When integrated into a service mesh, the OPA engine operates through a three-step workflow to resolve authorisation requests for the data intercepted by the sidecar proxies, as illustrated in Figure 8.1:

1. it accepts a JSON-formatted input that represents the attributes of the specific request;
2. it interprets the Rego rules to calculate a decision by correlating the input with the existing context and external data;
3. it delivers the evaluation outcome back to the requester as a JSON response, encapsulating the results of all applicable rules.

Despite Rego's flexibility, its stateless design presents some challenges for specific use cases. Because the engine cannot persist or modify data across different requests, it struggles to support paradigms that require memory of past events.

Currently, any necessary updates to the underlying data object must be handled manually by the service's application logic. This requirement forces security-related state management into the hands of the service developers, not maintaining a strict separation between policy specification and business logic, and increasing the risk of potential vulnerabilities, unsafe data usage, or other misconfigurations.

```

1 package app.rbac
2 import rego.v1
3
4 # By default, deny requests.
5 default allow := false
6
7 # Allow admins to do anything.
8 allow if user_is_admin
9
10 # Allow the action if the user is granted permission to perform
    the action.
11 allow if {
12     # Find grants for the user.
13     some grant in user_is_granted
14     # Check if the grant permits the action.
15     input.action == grant.action
16     input.type == grant.type
17 }
18
19 # user_is_admin is true if "admin" is among the user's roles as
    per data.user_roles
20 user_is_admin if "admin" in data.user_roles[input.user]
21
22 # user_is_granted is a set of grants for the user identified in
    the request.
23 # The 'grant' will be contained if the set 'user_is_granted' for
    every...
24 user_is_granted contains grant if {
25     # 'role' assigned an element of the user_roles for this
        user...
26     some role in data.user_roles[input.user]
27     # 'grant' assigned a single grant from the grants list for
        'role'...
28     some grant in data.role_grants[role]
29 }

```

Listing 8.1: Example of Rego policy for RBAC.

```

1 {"user": "alice", "action": "read", "object": "id123", "type": "dog"}

```

Listing 8.2: Input for the RBAC example.

```

1 {
2   "user_roles": {
3     "alice": ["admin"],
4     "bob": ["employee", "billing"],
5     "eve": ["customer"]
6   },
7   "role_grants": {
8     "customer": [{"action": "read", "type": "dog"}, [...]],
9     "employee": [{"action": "update", "type": "dog"}, [...]],
10    "billing": [{"action": "read", "type": "finance"}, [...]]
11  }
12 }

```

Listing 8.3: Data for the RBAC example.

```

1 {"allow":true, "user_is_admin":true, "user_is_granted":[]}

```

Listing 8.4: Output for the RBAC example for the provided input.

8.2 OWSM: OPA Wrapper State Manager

To overcome the “statelessness” of Rego, we introduce the *Open policy agent Wrapper State Manager* (OWSM), that is able to manage a dedicated state store to track persistent policy-relevant data, which the Rego engine can then query during evaluation. OWSM augments the OPA engine with native state management capabilities: by integrating it in their workflow, developers can offload the complexities of maintaining persistent authorisation data from the application’s business logic, ensuring that security-relevant state is handled entirely within the policy infrastructure.

The necessity for such a system is best illustrated by inherently stateful policies (see also Section 7.2), such as a token-based access control model for a service API. In this scenario, users are allocated a specific quota of tokens each month, and every API call must decrement the balance. A viable policy must both track the token expenses in real-time and reset the counters upon the start of a new billing cycle. Traditional policy engines cannot express this logic natively, as they lack the internal mechanisms required to modify and persist the “available tokens” variable across independent requests without external intervention.

Security requirements provide further evidence of the need for stateful management, particularly in scenarios explored in Casola et al. (2024), Verderame et al. (2023), and Verderame et al. (2025). Consider a system involving three services, *A*, *B*, and *C*, operating at distinct clearance levels, where *A* holds a higher classification than *C*. To prevent unauthorised data leakage from a high-security domain to a lower one, as outlined in the Bell-LaPadula security model, we must enforce a policy that prevents *B* from communicating with *C* if *B* has already engaged in communication with *A*. Notice that this pattern is very similar to the communication quarantine policy employed by ABCASTSM in Section 7.4.

Such a policy requires tracking the historical state of service interactions. Without a dedicated stateful framework, traditional policy engines are incapable of enforcing these constraints independently, once again forcing developers to integrate state-tracking into the services and blurring the line between security enforcement and application functionality.

Our architectural solution involves encapsulating the OPA engine within a custom API designed to interface with a dedicated datastore, enabling the system to maintain and modify persistent state at runtime, while maintaining compatibility with existing Rego policies.

Similarly to what happens in the generic service mesh implementation (Figure 7.1), the sidecar proxy interacts directly with an OWSM instance to determine whether a request should be authorised. To ensure scalability and fault tolerance, multiple OWSM instances can be deployed, typically one for each sidecar proxy, allowing for localised decision-making while maintaining access to a shared and synchronised state.

Requirement definitions. The core requirement of our architecture is the integration of primitives for reading from and writing to a persistent state that survives across multiple and independent policy queries. These primitives allow the decision engine to dynamically modify the data context upon which subsequent authorisations are based.

For the system to be effective, this state must remain accessible and consistent across all instances of the decision engine. In a typical service mesh environment, where engines are deployed as sidecars alongside

multiple server replicas, centralised state management is essential. For example, to enforce the limited-token policy, the token count must be synchronised globally. Without this uniform enforcement, a user could theoretically bypass the quota by cycling through different replicas, undermining the security policy's integrity.

To realise this architecture, the system is structured around two integrated components: a policy decision engine and a high-availability datastore. The choice to use OPA as the core engine is motivated by Rego's ability to generate arbitrary JSON objects as evaluation outputs. This flexibility allows us to embed state-update instructions directly within the authorisation response, simplifying the integration between the decision logic and the underlying state management layer.

Design. We avoid direct modifications to the OPA core, instead encapsulating it within a specialised wrapper API. This layer orchestrates the interaction between the policy engine and the stateful backend by synchronising the evaluation context with the persistent datastore. As illustrated in Figure 8.2, the system executes a transactional data flow:

1. the wrapper receives an authorisation query from a service and prepares OPA's evaluation context;
2. to prevent race conditions, the wrapper initiates a (pessimistic) lock on the relevant datastore records and retrieves the current state;
3. the retrieved state is injected into OPA as external data;
4. OPA evaluates the Rego rules, generating a response that includes both the authorisation decision (e.g. allow/deny) and the required state-update instructions;
5. the wrapper pushes the updated data back to the store and releases the lock, which is necessary to guarantee data integrity during concurrent policy evaluations;
6. the final authorisation result is returned to the requesting service, after the removal of the internal state metadata.

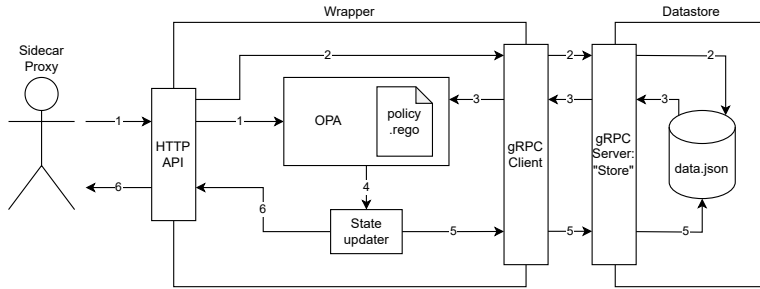


Figure 8.2: Diagram of OWSM implementation.

In our architecture, state modification is achieved through a reserved policy identifier, i.e. *state*. This policy is defined using standard Rego rules and produces a conventional JSON dictionary as its output. Upon evaluation, OWSM intercepts this JSON object and parses it to determine what needs to be persisted in the datastore. Only once the synchronisation is complete is the final authorisation decision delivered to the requesting service through the proxy.

Implementation. From an implementation perspective, we developed both the wrapper and the datastore using the Go programming language. This choice was suggested by the fact that OPA itself is written in Go and provides a native library that allows for interaction with its internal evaluation engine.

The datastore implements its external interface via gRPC, ensuring a strictly typed and high-performance communication contract between the wrapper components. Within this gRPC service, four specific procedures are defined to manage the lifecycle and retrieval of stateful information:

- **Get** to retrieve the current state;
- **Put** to update a specific key with a new value;
- **Lock** to ensure mutually exclusive access for concurrent clients;
- **Unlock** to release the lock in order to allow subsequent operations.

```

1 package tokencounter
2 import rego.v1
3
4 default allow := false
5
6 allow if {
7     input.user == "username"
8     data.counter > 0
9 }
10
11 state["counter"] := data.counter - 1 if allow

```

Listing 8.5: Rego code for the token counter example.

The wrapper communicates with this datastore through its gRPC service, while exposing a standard HTTP endpoint (specifically `/query`) to the user-developed service. This design mirrors the standard OPA HTTP API, ensuring that the transition to our stateful implementation remains transparent to existing services.

To validate the effectiveness of our system, we implemented both the “token counter” and the “three microservices” use cases. These scenarios were tested using OWSM and yielded a null error rate, empirically demonstrating that OWSM provides the enhanced expressiveness required for stateful logic. The implementation details and corresponding Rego logic for these cases are provided in Listings 8.5 and 8.6.

The OWSM implementation is available at CysecUD (2025).

8.3 Experimental evaluation

We conduct an experimental evaluation of OWSM to quantify its efficiency and the performance overhead it introduces relative to standard OPA. To ensure a fair and direct comparison, we focus on use cases that can be fully expressed in both environments, specifically, stateless policies that do not require persistent information. These benchmarks are derived from the representative examples found in the Access Control section of the Rego Playground (OpenPolicyAgent 2026). We evaluated OWSM using three distinct scenarios derived from industry-standard access control patterns.

```

1 package threemicroservices
2 import rego.v1
3
4 # B can talk to C until A talks to B
5 default allow := false
6 allow if {
7     input.source == "a"
8     input.dest == "b"
9 }
10 allow if {
11     input.source == "b"
12     input.dest == "c"
13     data.a_to_b == false
14 }
15 state["a_to_b"] if {
16     input.source == "a"
17     input.dest == "b"
18 }

```

Listing 8.6: Rego code for the three microservices example.

Use case 1: Role-Based Access Control This scenario implements a traditional RBAC model for the Pet Store API (Swagger 2026). The policy regulates how users view, adopt, or update pet records based on assigned roles. Access is granted by mapping users to roles, which in turn are linked to specific resource permissions;

Use case 2: Attribute-Based Access Control This model shifts the focus to attributes, while using the same API. Authorisation decisions are computed by evaluating characteristics associated with the user (e.g. title), the resource (e.g. pet type), and the environment, offering more granular control than static roles;

Use case 3: Hierarchical RBAC This case addresses the role explosion problem, i.e. the unmanageable growth of roles in large systems, by implementing a role hierarchy. Instead of creating unique roles for every possible permission set, roles are organised in a graph where child roles inherit permissions from their parents. When a request is submitted, the policy traverses this hierarchy to determine if the user's aggregate permissions satisfy the requirement.

8.3.1 Research questions and methodology

RQ1 [Time performance]. *What are the performance characteristics of the overhead introduced by OWSM during sequential request processing?*

It is important to quantify the impact of the OWSM layer on end-to-end latency. This investigation accounts for both the additional computational cycles required to wrap the OPA engine and the latency costs associated with orchestrating state modifications.

RQ2 [Access scalability]. *How does OWSM perform as the volume of simultaneous requests increases?*

Evaluating the system's behaviour under high concurrency is essential for determining its scalability within a distributed service mesh. This analysis focuses on how response times evolve as multiple, simultaneous requests compete for access to the centralised datastore.

To measure response times, we use Apache Benchmark (ab) version 2.4.62, a standard utility for web server benchmarking. We modified the ab source code to output timings in microseconds, bypassing the default rounding to the nearest millisecond. This was achieved by patching the `ap_round_ms` macro to return the raw runtime value (already recorded in microseconds) without altering the tool's core functionality.

To compare OPA and OWSM, we conduct experiments on both systems for every use case. Each session begins with a warmup phase of 10 non-concurrent requests, which prevents outlier high-latency values often caused by the initial system query.

To address RQ1, we execute batches of requests in sizes ranging from 100 to 102 400 (doubling at each step) for both OPA and OWSM: for each we determine the mean response time and the interquartile range (IQR).

For RQ2, we submit a total of 50 000 requests while varying the concurrency levels from 100 to 10 000 parallel requests. More precisely, the concurrency levels are: 100, 650, 1 200, 1 750, 2 300, 2 850, 3 400, 3 950, 4 500, 5 050, 5 600, 6 150, 6 700, 7 250, 7 800, 8 350, 8 900, 9 450, and 10 000.

We then calculate the mean and IQR of the response times for each of these concurrency settings.

For reproducibility, the experiments are conducted on a server running Debian GNU/Linux 12 (kernel 6.10-28), equipped with an Intel(R) Core(TM) i9-10900 CPU @ 2.80GHz (20 threads) and 128 GB of RAM.

Threats to validity. To address inherent measurement randomness, we executed multiple requests for each configuration and evaluated the statistical significance of the differences between the response time IQRs of OPA and OWSM using the Mann-Whitney U test: this non-parametric test determines if one sample group consistently yields higher values than the other. We calculated the U statistic and the associated p-value to verify if the performance differences were significant at a level of $\alpha = 0.05$.

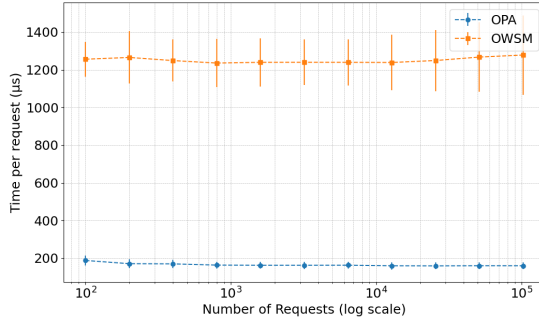
Furthermore, we supported our trend analysis by fitting a linear regression model to the results and evaluating the coefficient of determination (R^2). To minimise threats to external validity and ensure the generalizability of our findings, we utilised three diverse, representative use cases sourced directly from the official OPA playground.

8.3.2 Results

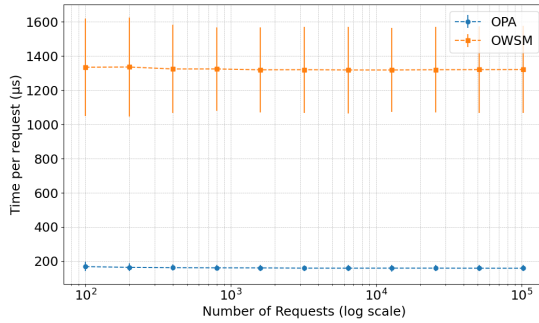
Research Question 1. Figures 8.3a to 8.3c illustrate the performance overhead OWSM introduces compared to native OPA across the three stateless use cases. Some variability in individual query response times arises because OWSM relies on two distinct components (the wrapper and the datastore) communicating via an API. This architectural split is reflected in OWSM’s significantly higher IQR, with a p-value < 0.05 .

Despite this variability, the overhead remains nearly constant across all measured points ($|\text{slope}| < 0.001$, p-value > 0.05). This is a positive result, indicating that OWSM’s overhead does not compound as the sequence of consecutive requests grows. A stable overhead ensures that OWSM can process long sequences of requests without accumulating delays, providing the predictability and reliability essential for real-world applications involving repeated queries.

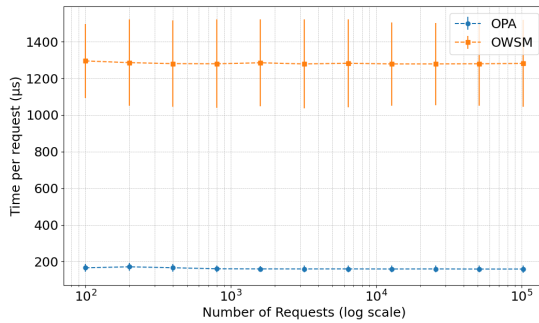
***RQ1 answer:** OWSM introduces a nearly constant overhead of approximately 1 ms, which remains stable throughout the entire sequence of requests.*



(a) Time performance (use case 1)

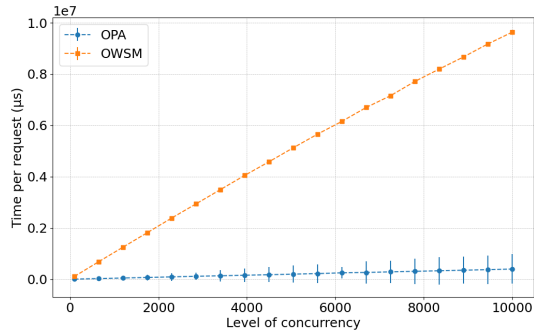


(b) Time performance (use case 2)

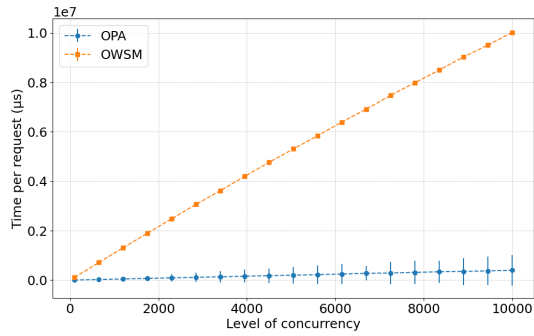


(c) Time performance (use case 3)

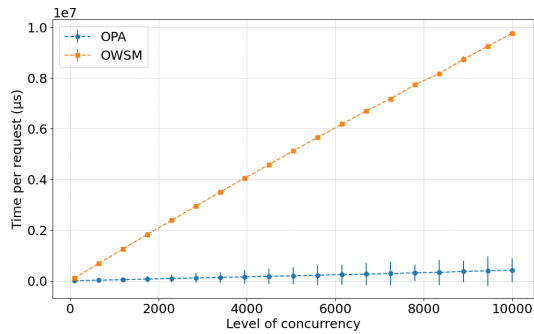
Figure 8.3: Comparison between OPA and OWSM: time performance.



(a) Concurrent scalability (use case 1)



(b) Concurrent scalability (use case 2)



(c) Concurrent scalability (use case 3)

Figure 8.4: Comparison between OPA and OWSM: concurrency scalability.

Research Question 2. Figures 8.4a to 8.4c demonstrate that the overhead OWSM introduces for concurrent requests follows a strictly linear trend ($R^2 > 0.999$, p-value < 0.05). This behaviour originates from the datastore's locking mechanism, which serialises request processing even when no state modifications occur. These findings align with previous results; for instance, at a concurrency level of 10 000, the elapsed time per request is approximately 10 000 milliseconds.

Notably, OWSM exhibits significantly more predictable response times than OPA, characterised by a much lower IQR. Furthermore, OWSM's mean response time is roughly 10 times lower than that of OPA, demonstrating its superior consistency when managing high volumes of concurrent requests.

***RQ2 answer:** As concurrency levels rise, OWSM response times increase linearly while maintaining significantly higher temporal stability.*

Chapter 9

Conclusions

The shift toward heterogeneous distributed ecosystems has fundamentally altered the landscape of protocol security. As established in the beginning of this work, traditional “all-or-nothing” security assessments are no longer sufficient to protect systems where trust assumptions may be only partially violated. This thesis has addressed this challenge by moving away from monolithic entity modelling in favour of a fine-grained, interface-based formal framework.

By treating communication interfaces as the primary units of both compromise and composition, we have provided a methodology that remains robust across diverse architectures. This chapter summarises how our research has navigated the complexities of both coupled and decoupled systems, the integration of service mesh logic into formal models, and the implementation of stateful policy enforcement. Together, these contributions offer a formally sound, rigorous, thorough, and extensible framework for securing the next generation of distributed systems.

9.1 Contributions summary

RQ1 and RQ2: Formalising fine-grained adversaries in coupled and uncoupled systems. To address the challenges of verifying complex, multi-actor communication, we introduced a *formal framework* designed

for the *symbolic verification* of both *coupled* and *decoupled* protocols. This methodology is intentionally general as it focuses on the security implications of what happens when specific interfaces are compromised, effectively abstracting away the specifics of how that control was acquired.

This abstraction allows for broad threat modelling beyond traditional sniffing or message forging. For instance, a malware infection granting root access to a device can be modelled as an attacker fully controlling all of that actor's interfaces. Similarly, a side-channel attack that allows an adversary to capture plaintext data flowing between hardware components can be modelled as the attacker acquiring read-only access to the specific interface connecting those.

Unlike analyses using the traditional Dolev-Yao model, that often assume a binary state where agents and channels are either fully honest or fully compromised, our approach accounts for an adversary with *interface-level access*, allowing a *fine-grained* analysis of diverse threat scenarios. To maintain computational efficiency during automated analysis, and thus reducing verification time, we also developed strategies to prune the state space without sacrificing soundness, ensuring the system remains viable for large-scale analysis.

A primary advantage of this proposed methodology is its *future-proof* and *compositional* design: it accounts for novel vulnerabilities, provided their impact manifests as a compromise of the interfaces defined within the model. If new attackable interfaces are discovered, the system can be updated without invalidating previous verification results, since already analysed scenarios can be interpreted within the new architecture as simply assuming the security of the newly added interfaces.

For *publish-subscribe* environments, we extended the standard Dolev-Yao attacker model to account for decoupled communication patterns. We specifically analysed the criticality of the head and tail set, demonstrating how an attacker's influence shifts significantly depending on whether they compromise a publisher's output interface or a subscriber's input interface in read or write. This distinction is important, since the point of entry dictates whether an attacker gains a localised or a global view of the system's data exchange.

We applied this methodology to verify the *Italian CIEId system*, providing a complete formalisation of the Level 2 and Level 3 authentication schemes using both sequence diagrams and ProVerif code. By carrying out an extensive analysis under thousands of possible attack scenarios, we exposed specific security vulnerabilities within the L2SMS, L2PSH, and L3APP protocols. Based on these findings, we proposed targeted *improvements* to bring their security on par with more robust schemes like L2QRC. Our analysis also highlighted that L3APP possesses a broader attack surface than L3DSK and is thus more attackable.

A significant observation from our analysis is that certain channels may appear non-essential for the primary security goals of a protocol. For example, the security of the mobile TLS channel is not strictly required for the safety of the L2QRC protocol. However, we maintain that data should never be transmitted unprotected. Minimising the information accessible to an attacker is a crucial *defence-in-depth* strategy, preventing the adversary from gaining additional intelligence that could lead to unforeseen exploits. This principle of information minimisation remains a cornerstone of our security recommendations.

Finally, the methodology followed in this work is highly *adaptable* and can be applied to the multi-factor authentication schemes of other eIDAS-compliant digital identity cards using the provided scripts and artefacts (Paier et al. 2025).

RQ3: Formalising service mesh architectures and sidecar proxy logic in our model. We addressed the architectural complexity of *service meshes* by developing a method to represent sidecar proxy logic within our formal verification framework. The primary challenge in modelling these systems lies in the potential for state-space explosion; a naïve modelling approach, where each sidecar is represented as a standalone process running in parallel with its corresponding application service, effectively doubles the number of active processes.

Because ProVerif must evaluate all possible interleavings of process actions to identify attack traces, this often prevents the solver from reaching a conclusion. To overcome this, we utilised the m4 macro processor to

perform in-place expansion, compiling the service mesh logic directly into the protocol’s communication primitives during a pre-processing phase. By injecting security logic into the inputs and outputs of the system, we maintain the original process count and eliminate uninteresting internal communication hops, ensuring the model remains small enough for efficient automated verification while accurately capturing the complex behaviour of the data plane.

Our results confirm that offloading security logic to a functional sidecar is a mathematically viable solution for hardening protocols against the fine-grained attackers defined by our threat model. For example, we can demonstrate this by implementing a service mesh which enforces the use of cryptographic signatures for data integrity and using it to patch attacks identified in a vulnerable simple protocol. In this architecture, the output macro encapsulates outgoing data by appending a cryptographic signature generated from a private key, while the corresponding input macro intercepts the traffic to verify the signature before it ever reaches the application logic.

Note that this setup ensures that application processes remain entirely decoupled from sensitive security material: since they operate without direct knowledge of cryptographic secrets or signing keys, again we achieve a defence-in-depth posture where even byzantine or compromised nodes cannot forge authentication tokens or leak keys they never possessed.

Furthermore, our framework extends beyond signatures to support the representation of *stateful policies*. By designing m4 macros that interact with ProVerif’s tables, we enable the model to write to data cells and verify system status based on the flow of incoming or outgoing messages. This allows us to model complex invariants, such as temporal constraints or quantitative bounds, without modifying the core business logic of the protocol: we can write the protocol logic as a simple sequence of steps while the service mesh macros ensure that the execution adheres to the required stateful invariants. This separation of concerns allows the formal model to prove that the global integrity and secrecy of a system can be preserved even when individual component logic behaves unpredictably or is subject to external compromise.

RQ4: Integrating stateful policy languages into service mesh architectures. To address the limitations of traditional stateless models, we explored the integration of specialised *policy languages* capable of managing changing protocol state. While the Open Policy Agent (OPA) has become widely used for authorisation, its native language, Rego, is fundamentally stateless. This is problematic when attempting to enforce policies that depend on historical context or temporal invariants: to overcome this “statelessness” we introduced the Open policy agent Wrapper State Manager (OWSM). It encapsulates the OPA engine within a custom API designed to interface with a dedicated datastore, allowing the system to maintain and modify persistent state at runtime while maintaining full compatibility with existing Rego policies.

The architectural viability of OWSM was confirmed through a series of evaluations focused on functional sanity and performance overhead. Our implementation follows the generic service mesh pattern where each sidecar proxy interacts directly with an OWSM instance to resolve authorisation requests. To ensure scalability and fault tolerance in distributed environments, we demonstrated that multiple OWSM instances can be deployed to allow for localised decision-making while safely accessing the shared state.

Our experimental results quantify the *performance overhead* introduced by OWSM’s expanded functionality relative to standard OPA. The stability of this overhead across long request sequences, alongside the predictable performance under heavy concurrent loads, indicates that OWSM can be a solution for addressing OPA’s lack of native state management primitives. It is important to notice that the current prototype’s locking mechanism only serves as a base for future optimisation: more sophisticated concurrency control strategies to further optimise throughput and minimise latency in highly distributed environments can be adopted.

We have released OWSM’s implementation at CysecUD (2025) as an open-source project under the *MIT license*. This release provides the research community with a functional codebase to integrate stateful policy management into their own software stacks, easing the adoption of state-aware security invariants in diverse microservice environments.

9.2 Future work

The contributions presented in this thesis establish the groundwork for a more rigorous and automated approach to protocol security, also using service meshes. However, the complexity of modern distributed systems offers several hints for further research, ranging from hardware-level threat modelling to the formalisation of stateful policy languages.

While our interface-based model provides a detailed abstraction of protocol vulnerabilities, there is significant potential in extending this framework to address hardware-specific threats. Side-channel attacks, such as those targeting timing, power consumption, or electromagnetic emissions, remain a substantial challenge for secure elements like smart cards. While hardware-level vulnerabilities were not the primary focus of this analysis, our interface model is designed for extensibility in this direction. For instance, a side-channel attack leaking a private key could be formalised by introducing an isolated, read-only interface to the card’s internal state. Similarly, fault-injection attacks could be modelled as write-only control over specific input interfaces. We reserve the comprehensive formalisation of these hardware extensions for future investigation.

Beyond the authentication phase, a complete security analysis must also encompass the full system lifecycle, particularly the enrolment and recovery procedures. These phases are often the weakest link in Multi-Factor Authentication infrastructures, involving distinct security principals and interactions that differ from the core authentication exchange. Currently, a formal analysis of these processes for systems like the CIE is hindered by a lack of public documentation regarding the underlying administrative procedures: future work should focus on securing them. Furthermore, a thorough investigation of the official CIE implementation to ensure strict adherence to security best practices and the minimisation of potential attack vectors is much needed.

Since our methodology abstracts away from the specific analysis tool, a natural extension is to evaluate other symbolic backends, such as Tamarin (Meier et al. 2013). The two systems embody complementary verification paradigms: ProVerif follows a deductive approach, translating the

protocol into a set of Horn clauses that are discharged by an automatic resolution procedure, whereas Tamarin is based on multiset rewriting and natively supports unbounded mutable state, inductive reasoning, and interactive proof construction. We adopted ProVerif because its fully automatic verification and efficiency suit our setting, in which the scenario-generation process yields a large number of protocol variants to be verified. Moreover, the additional expressiveness offered by Tamarin was not required by our analyses: the limited stateful behaviour we needed, i.e. the locks of the ABCAST example, is adequately captured by ProVerif tables, which can be used to model synchronisation mechanisms in decoupled systems.

A promising way to decouple the methodology from any single back-end is to adopt SAPIC+ (Cheval et al. 2022), a protocol-specification front-end based on a stateful applied π -calculus that transpiles natively to both ProVerif and Tamarin. A single SAPIC+ model could thus be verified across multiple tools, combining ProVerif’s efficiency with Tamarin’s expressiveness whenever a particular scenario demands it, while gaining additional confidence from the cross-validation of independent results.

Perhaps the most significant area for future development lies in the formalisation of the policy evaluation process. Currently, neither Rego nor our OWSM prototype provides a rigorous and clearly defined formal semantics for their evaluation effects, thus complicating the systematic verification of complex security policies. To address this, we plan to transition our stateful architecture toward Cedar (Cutler et al. 2024), an open-source authorisation language that features a robust typing system and a formally defined semantics. This rigour is critical, as it enables the formal proof of language properties and supports the verification of the engine’s implementation to reduce the trusted code base.

To accomplish this, we are currently developing Strobilus (Baldo et al. 2026), a specialised language designed to define effects over policy-specific data. Like OWSM with Rego, Strobilus is engineered to seamlessly integrate with Cedar, allowing developers to implement stateful logic without modifying Cedar’s core syntax or evaluation engine. Unlike OWSM, Strobilus has a formal semantics, a robust typing system, and

a guarantee of termination. Our initial Rust-based prototype suggests that this architecture significantly outperforms external methods for managing policy data. By providing a secure, verifiable solution for both stateless and stateful authorisation, this approach aims to fully realise the potential of the policy-as-code paradigm in highly distributed and high-performance environments.

Bibliography

- Abadi, Martin and Phillip Rogaway (2002). “Reconciling two views of cryptography (the computational soundness of formal encryption)”. In: *Journal of cryptology* 15.2, pp. 103–127.
- AGID (2026). *Manuale Tecnico per gli erogatori di servizi pubblici e privati*. Available at <https://docs.italia.it/italia/cie/cie-manuale-tecnico-docs/it/master/overview.html>.
- Alaca, Furkan and P. C. van Oorschot (2016). “Device fingerprinting for augmenting web authentication: classification and analysis of methods”. In: *Proc. 32nd Conference on Computer Security Applications*. ACM, pp. 289–301. ISBN: 9781450347716.
- Alboqmi, Rami, Sharmin Jahan, and Rose F Gamble (2024). “A risk adaptive access control model for the service mesh architecture”. In: *2024 IEEE 3rd International Conference on Computing and Machine Intelligence (ICMI)*. IEEE, pp. 1–6.
- Aziz, Benjamin (2016). “A formal model and analysis of an IoT protocol”. In: *Ad Hoc Networks* 36, pp. 49–57.
- Baldo, Massimiliano, Pietro Di Gianantonio, Matteo Paier, and Marino Miculan (2026). “Strobilus: Enriching Cedar with Stateful Policies”. In: *Proceedings of the 31st ACM Symposium on Access Control Models and Technologies*. SACMAT '26. Canada: Association for Computing Machinery, pp. 205–216. DOI: 10.1145/3750555.3811892.
- Baldo, Massimiliano, Fabio Ionut Ion, Marino Miculan, Matteo Paier, and Vincenzo Riccio (2025). “OWSM: Empowering Rego for Stateful Access Control”. In: *Proceedings of the Joint National Conference on Cybersecurity (ITASEC & SERICS 2025), Bologna, Italy, February 03-08, 2025*. Vol. 3962. CEUR Workshop Proceedings. CEUR-WS.org. URL: <https://ceur-ws.org/Vol-3962/paper49.pdf>.

- Balliu, Musard, Massimo Merro, Michele Pasqua, and Mikhail Shcherbakov (2021). “Friendly fire: cross-app interactions in IoT platforms”. In: *ACM Transactions on Privacy and Security (TOPS)* 24.3, pp. 1–40.
- Barbosa, Manuel, Alexandra Boldyreva, Shan Chen, and Bogdan Warinschi (2021). “Provable security analysis of FIDO2”. In: *Annual International Cryptology Conference*. Springer, pp. 125–156.
- Baresi, Luciano, Giorgio Gerosa, Carlo Ghezzi, and Luca Mottola (2007). “Playing with time in publish-subscribe using a domain-specific model checker”. In: *Proceedings of the 2007 conference on Specification and verification of component-based systems: 6th joint meeting of the European conference on software engineering and the ACM SIGSOFT symposium on the foundations of software engineering*, pp. 55–62.
- Baresi, Luciano, Carlo Ghezzi, and Luca Mottola (2006). “Towards fine-grained automated verification of publish-subscribe architectures”. In: *International Conference on Formal Techniques for Networked and Distributed Systems*. Springer, pp. 131–135.
- (2010). “Loupe: Verifying publish-subscribe architectures with a magnifying lens”. In: *IEEE Trans. on Software Engin.* 37.2, pp. 228–246.
- Bartocci, Ezio, Thomas Ferrère, Thomas A Henzinger, Dejan Nickovic, and Ana Oliveira Da Costa (2022). “Information-flow interfaces”. In: *International Conference on Fundamental Approaches to Software Engineering*. Springer, pp. 3–22.
- Basin, David, Cas Cremers, and Catherine Meadows (2018). “Model checking security protocols”. In: *Handbook of Model Checking*. Springer, pp. 727–762.
- Bhargavan, Karthikeyan, Bruno Blanchet, and Nadim Kobeissi (2017). “Verified models and reference implementations for the TLS 1.3 standard candidate”. In: *2017 IEEE Symposium on Security and Privacy (S&P)*, pp. 483–502.
- Bhargavan, Karthikeyan, Cédric Fournet, Ricardo Corin, and Eugen Zalinescu (2008). “Cryptographically verified implementations for TLS”. In: *Proceedings of the 15th ACM conference on computer and Communications security*, pp. 459–468.
- Birman, Kenneth P and Thomas A Joseph (1987). “Reliable communication in the presence of failures”. In: *ACM Transactions on Computer Systems (TOCS)* 5.1, pp. 47–76.
- Blanchet, Bruno (2008). “A computationally sound mechanized prover for security protocols”. In: *IEEE Transactions on Dependable and Secure Computing* 5.4, pp. 193–207.

- Blanchet, Bruno et al. (2016). “Modeling and verifying security protocols with the applied pi calculus and ProVerif”. In: *Foundations and Trends in Privacy and Security* 1.
- Blanchet, Bruno (2025). *ProVerif users - Research papers*. Available at <https://bblanche.gitlabpages.inria.fr/proverif/proverif-users.html>.
- Bolignano, Dominique (1996). “An approach to the formal verification of cryptographic protocols”. In: *Proceedings of the 3rd ACM Conference on Computer and Communications Security*, pp. 106–118.
- Burrows, Michael, Martin Abadi, and Roger Needham (1990). “A logic of authentication”. In: *ACM Trans. on Computer Systems* 8.1, pp. 18–36.
- Casola, Valentina, Vincenzo Riccio, Giuseppe Tricomi, Giovanni Merlino, Pietro Di Gianantonio, Bruno Crispo, Massimiliano Rak, and Antonio Puliafito (2024). “SecCO-OC: securing microservice-base apps”. In: *Proc. 10th Italian Conference on ICT for Smart Cities and Communities*.
- Chandramouli, Ramaswamy, Zack Butcher, Aradhna Chetal, et al. (2021). “Attribute-based access control for microservices-based applications using a service mesh”. In: *NIST Special Publication* 800.S 41.
- Cheval, Vincent, Charlie Jacomme, Steve Kremer, and Robert Künnemann (2022). “{SAPIC+}: protocol verifiers of the world, unite!” In: *31st USENIX Security Symposium (USENIX Security 22)*, pp. 3935–3952.
- Chien, Hung-Yu, An-Tong Shih, and Yuh-Ming Huang (2025). “Exploring MQTT Broker-Based, End-to-End Models for Security and Efficiency”. In: *Sensors* 25.17, p. 5308.
- Cortier, Véronique, Steve Kremer, and Bogdan Warinschi (2011). “A survey of symbolic methods in computational analysis of cryptographic systems”. In: *Journal of Automated Reasoning* 46.3, pp. 225–259.
- Costa, Gabriele, Silvia De Francisci, and Rocco De Nicola (2025). “The Beauty and the Beast: A survey on process algebras and cybersecurity”. In: *Computer Science Review* 57, p. 100758.
- Cutler, Joseph W., Craig Disselkoen, Aaron Eline, Shaobo He, Kyle Headley, Michael Hicks, Kesha Hietala, Eleftherios Ioannidis, John Kastner, Anwar Mamat, et al. (2024). “Cedar: A New Language for Expressive, Fast, Safe, and Analyzable Authorization”. In: *Proceedings of the ACM on Programming Languages* 8.OOPSLA1, pp. 670–697.
- CysecUD (2025). *OWSM: OPA Wrapper State Manager*. <https://github.com/cysecud/owsm>.
- DDS Foundation (2025). *Data Distribution Service Foundation*. Available at <https://www.dds-foundation.org>.

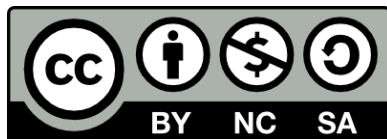
- De Alfaro, Luca and Thomas A Henzinger (2001). "Interface theories for component-based design". In: *International Workshop on Embedded Software*. Springer, pp. 148–165.
- Delzanno, Giorgio (2019). "Towards the Automated Verification of Publish/Subscribe Networks." In: *OVERLAY@ AI* IA*, pp. 35–40.
- Diffie, Whitfield and Martin E. Hellman (1976). "New Directions in Cryptography". In: *IEEE Trans. on Information Theory* 22.6, pp. 644–654.
- Disselkoben, Craig, Aaron Eline, Shaobo He, Kyle Headley, Michael Hicks, Kesha Hietala, John Kastner, Anwar Mamat, Matt McCutchen, Neha Rungta, et al. (2024). "How we built Cedar: A verification-guided approach". In: *Proc. 32nd ACM International Conference on the Foundations of Software Engineering*, pp. 351–357.
- Dolev, Danny and Andrew Yao (1983). "On the security of public key protocols". In: *IEEE Transactions on information theory* 29.2, pp. 198–208.
- Esparza-Pedro, Javier, Francesc D Muñoz-Escóí, and José M Bernabéu-Aubán (2024). "Modeling microservice architectures". In: *Journal of Systems and Software* 213, p. 112041.
- Ganguli, Mrityika, Sunku Ranganath, Subhiksha Ravisundar, Abhirupa Layek, Dakshina Ilangovan, and Edwin Verplanke (2021). "Challenges and Opportunities in Performance Benchmarking of Service Mesh for the Edge". In: *2021 IEEE International Conference on Edge Computing (EDGE)*, pp. 78–85. DOI: 10.1109/EDGE53862.2021.00020.
- Gritzalis, Stefanos, Diomidis Spinellis, and Panagiotis Georgiadis (1999). "Security protocols over open networks and distributed systems: Formal methods for their analysis, design, and verification". In: *Computer Communications* 22.8, pp. 697–709.
- Hirsch, Dan, Paola Inverardi, and Ugo Montanari (1998). "Graph grammars and constraint solving for software architecture styles". In: *Proc. of the third international workshop on Software architecture*, pp. 69–72.
- Holgers, Tobias, David E Watson, and Steven D Gribble (2006). "Cutting through the Confusion: A Measurement Study of Homograph Attacks." In: *USENIX Annual Technical Conference*, pp. 261–266.
- Ion, Mihaela (2013). "Security of publish/subscribe systems". PhD Dissertation. Trento, Italy: University of Trento.
- Jacomme, Charlie and Steve Kremer (2021). "An extensive formal analysis of multi-factor authentication protocols". In: *ACM Transactions on Privacy and Security* 24, pp. 1–34.
- Kernighan, Brian and Dennis Ritchie (1977). *The m4 macro processor*. Tech. rep. Bell Laboratories Murray Hill, NJ.

- Khelifi, Siwar, Ahmed Hadj Kacem, et al. (2008). "Specification and verification of the structural and behavioral properties of publish/subscribe architectures". In: *Second international workshop on verification and evaluation of computer and communication systems (VECoS 2008)*.
- Kim, Hwimin, Dae-Kyoo Kim, and Alaa Alaerjan (2021). "ABAC-based security model for DDS". In: *IEEE Transactions on Dependable and Secure Computing* 19.5, pp. 3113–3124.
- Kobeissi, Nadim, Karthikeyan Bhargavan, and Bruno Blanchet (Apr. 2017). "Automated Verification for Secure Messaging Protocols and Their Implementations: A Symbolic and Computational Approach". In: *2nd IEEE European Symposium on Security and Privacy*. Paris, France, pp. 435–450. DOI: 10.1109/EuroSP.2017.38. URL: <https://inria.hal.science/hal-01575923>.
- Larisch, James, Timothy Alberdingk Thijm, Suleman Ahmad, Peter Wu, Tom Arnfeld, and Marwan Fayed (2024). "Topaz: Declarative and Verifiable Authoritative DNS at CDN-Scale". In: *Proceedings of the ACM SIGCOMM 2024 Conference*, pp. 891–903.
- Le Métayer, D. (1998). "Describing software architecture styles using graph grammars". In: *IEEE Transactions on Software Engineering* 24.7, pp. 521–533. DOI: 10.1109/32.708567.
- Li, Li, Jun Sun, Yang Liu, Meng Sun, and Jin-Song Dong (2017). "A formal specification and verification framework for timed security protocols". In: *IEEE Transactions on Software Engineering* 44.8, pp. 725–746.
- Li, Wubin, Yves Lemieux, Jing Gao, Zhuofeng Zhao, and Yanbo Han (2019). "Service Mesh: Challenges, State of the Art, and Future Research Opportunities". In: *2019 IEEE Int. Conf. on Service-Oriented System Engineering (SOSE)*, pp. 122–1225. DOI: 10.1109/SOSE.2019.00026.
- Linkerd (2024). *The world's most advanced service mesh*. Available at <https://linkerd.io/>.
- Liu, Yanan, Yong Guan, Xiaojuan Li, Rui Wang, and Jie Zhang (2018). "Formal analysis and verification of DDS in ROS2". In: *2018 16th ACM/IEEE International Conference on Formal Methods and Models for System Design (MEMOCODE)*. IEEE, pp. 1–5.
- Lorch, Markus, Seth Proctor, Rebekah Lepro, Dennis Kafura, and Sumit Shah (2003). "First experiences using XACML for access control in distributed systems". In: *Proc. 2003 ACM workshop on XML security*, pp. 25–37.
- Margheri, Andrea, Massimiliano Masi, Rosario Pugliese, and Francesco Tiezzi (2019). "A Rigorous Framework for Specification, Analysis and

- Enforcement of Access Control Policies". In: *IEEE Trans. Software Eng.* 45.1, pp. 2–33.
- Meadows, Catherine (1992). "Applying formal methods to the analysis of a key management protocol". In: *Journal of Computer security* 1.1, pp. 5–35.
- Meier, Simon, Benedikt Schmidt, Cas Cremers, and David Basin (2013). "The TAMARIN prover for the symbolic analysis of security protocols". In: *International conference on computer aided verification*. Springer, pp. 696–701.
- Miculan, Marino and Nicola Vitacolonna (2021). "Automated Symbolic Verification of Telegram's MTPROTO 2.0". In: *Proceedings of the 18th International Conference on Security and Cryptography, SECRYPT 2021*. Ed. by Sabrina De Capitani di Vimercati and Pierangela Samarati. SCITEPRESS, pp. 185–197. DOI: 10.5220/0010549601850197.
- (2023). "Automated verification of Telegram's MTPROTO 2.0 in the symbolic model". In: *Comput. Secur.* 126, p. 103072. DOI: 10.1016/J.COSE.2022.103072.
- Mitchell, John C, Mark Mitchell, and Ulrich Stern (1997). "Automated analysis of cryptographic protocols using mur/spl phi". In: *Proceedings. 1997 IEEE Symposium on Security and Privacy (Cat. No. 97CB36097)*. IEEE, pp. 141–151.
- Oku, Reiya, Kohei Shiimoto, and Yoshihiro Ohba (2022). "Decentralized Identifier and Access Control Based Architecture for Privacy-Sensitive Data Distribution Service". In: *2022 IEEE 8th World Forum on Internet of Things (WF-IoT)*, pp. 1–6. DOI: 10.1109/WF-IoT54382.2022.10152128.
- OpenFGA (2024). *Relationship-based access control made fast, scalable, and easy to use*. Available at <https://openfga.dev/>.
- OpenPolicyAgent (2024). *Rego documentation*. Available at <https://www.openpolicyagent.org/docs/latest/policy-language/>.
- (2026). *The Rego playground*. Web application available at <https://play.openpolicyagent.org/>.
- Orlovsky, Arie and Danny Raz (2007). "Decentralized enforcement of security policies for distributed computational systems". In: *Proceedings of the 2007 ACM symposium on Applied computing*, pp. 241–248.
- Paier, Matteo, Roberto Van Eeden, and Marino Miculan (2024). "Formal Analysis of Multi-Factor Authentication Schemes in Digital Identity Cards". In: *Software Engineering and Formal Methods - 22nd International Conference, SEFM 2024, Aveiro, Portugal, November 6-8, 2024, Proceedings*.

- Vol. 15280. Lecture Notes in Computer Science. Springer, pp. 423–440. DOI: 10.1007/978-3-031-77382-2_24.
- (May 2025). *Formal Modelling and Verifying eIDAS Multi-Factor Authentication with Interface-Based Threat Analysis – Artefact*. DOI: 10.5281/zenodo.15403641.
 - (2026). “Formal modelling and verifying eIDAS multi-factor authentication with interface-based threat analysis”. In: *Software and Systems Modeling*. DOI: 10.1007/s10270-026-01375-9.
- Pallewatta, Samodha and Muhammad Ali Babar (2024). “Towards secure management of edge-cloud IoT microservices using policy as code”. In: *European Conference on Software Architecture*. Springer, pp. 270–287.
- Pang, Ruoming et al. (2019). “Zanzibar: Google’s Consistent, Global Authorization System”. In: *2019 USENIX Annual Tech. Conf. (ATC ’19)*.
- Paul, Alen, Rishi Manoj, and S. Udhayakumar (2024). “Amazon Web Services Cloud Compliance Automation with Open Policy Agent”. In: *2024 International Conf. on Expert Clouds and Applications (ICOECA)*. IEEE, pp. 313–317. DOI: 10.1109/ICOECA62351.2024.00063.
- Robby, Matthew B Dwyer, and John Hatcliff (2003). “Bogor: an extensible and highly-modular software model checking framework”. In: *ACM SIGSOFT Software Engineering Notes* 28.5, pp. 267–276.
- ROS (2025). *ROS - Robot Operating System*. Available at <https://www.ros.org/>.
- Sappa, Ankita (2025). “Context-aware security policies for data-driven applications in cloud-native architectures”. In: *Journal of Internet Services and Information Security* 15.2, pp. 43–64.
- Sheikh, Ozair, Serjik Dikaleh, Dharmesh Mistry, Darren Pape, and Chris Felix (2018). “Modernize digital applications with microservices management using the Istio service mesh”. In: *CASCON ’18: Proceedings of the 28th Annual International Conference on Computer Science and Software Engineering*. IBM, pp. 359–360.
- Swagger (2026). *Swagger Petstore - OpenAPI 3.0*. Available at <https://petstore3.swagger.io/>.
- Syverson, Paul F and Paul C Van Oorschot (1994). “On unifying some cryptographic protocol logics”. In: *Proceedings of 1994 IEEE computer society symposium on research in security and privacy*. IEEE, pp. 14–28.
- Ullah, Kaleem, Imran Rashid, Hammad Afzal, Mian Muhammad Waseem Iqbal, Yawar Abbas Bangash, and Haider Abbas (2020). “SS7 vulnerabilities—a survey and implementation of machine learning vs rule based filtering for detection of SS7 network attacks”. In: *IEEE Communications Surveys & Tutorials* 22.2, pp. 1337–1371.

- Van Eeden, Roberto, Matteo Paier, and Marino Miculan (2024). "A Formal Analysis of CIE Level 2 Multi-Factor Authentication via SMS OTP". In: *Proceedings of the 21st International Conference on Security and Cryptography, SECRIPT 2024, Dijon, France, July 8-10, 2024*. SCITEPRESS, pp. 483–491. DOI: 10.5220/0012768300003767.
- Varadharajan, Vijay (1989). "Verification of network security protocols". In: *Computers & Security* 8.8, pp. 693–708.
- Verderame, Luca, Luca Caviglione, Roberto Carbone, and Alessio Merlo (2023). "SecCo: Automated Services to Secure Containers in the DevOps Paradigm". In: *Proc. 2023 International Conference on Research in Adaptive and Convergent Systems, RACS 2023*. ACM, 10:1–6. DOI: 10.1145/3599957.3606222. URL: <https://doi.org/10.1145/3599957.3606222>.
- Verderame, Luca et al. (2025). "From Edge to Cloud: Securing Distributed Containerized Applications". In: *2025 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)*, pp. 619–629. DOI: 10.1109/EuroSPW67616.2025.00077.
- Wang, Ding, Debiao He, Ping Wang, and Chao-Hsien Chu (2014). "Anonymous two-factor authentication in distributed systems: Certain goals are beyond attainment". In: *IEEE Transactions on Dependable and Secure Computing* 12.4, pp. 428–442.
- White, T., M. N. Johnstone, and M. Peacock (2017). "An investigation into some security issues in the DDS messaging protocol". In: *The Proceedings of 15th Australian Information Security Management Conference*. Ed. by C. Valli. Perth, Western Australia: Edith Cowan University, pp. 132–139.
- Zhou, Yu, Jidong Ge, Pengcheng Zhang, and Weigang Wu (2016). "Model based verification of dynamically evolvable service oriented systems". In: *Science China Information Sciences* 59.3, p. 32101.



Unless otherwise expressly stated, all original material of whatever nature created by Matteo Paier and included in this thesis, is licensed under a Creative Commons Attribution Noncommercial Share Alike 3.0 Italy License.

Check on Creative Commons site:

<https://creativecommons.org/licenses/by-nc-sa/3.0/it/legalcode/>

<https://creativecommons.org/licenses/by-nc-sa/3.0/it/deed.en>

Ask the author about other uses.