Indian Institute of Science
भारतीय विज्ञान संस्थान

Enabling Secure and Scalable Data Analytics on Edge Devices

Journal:	*IISc - Ph.D Thesis Processing*
Thesis ID	IISc-2025-0359.R3
Manuscript Type:	Synopsis and Thesis
Date Submitted by the Author:	19-Jul-2026
Complete List of Authors:	JOSEPH, ARUN; IISC Bangalore, CSA
Keywords:	Privacy preserving video analytics, Trusted execution environments, Multiparty computation

Thesis Synopsis

Indian Institute of Science Bangalore

Title: Enabling Secure and Scalable Data Analytics on Edge Devices

Closed-Circuit Television (CCTV) systems play a vital role in public safety, urban monitoring, and criminal investigation by providing continuous visual surveillance. With the increasing urbanization and affordability of cameras, cities worldwide are witnessing an exponential rise in camera density-for instance, Delhi, India, now hosts approximately 1,446 cameras per square mile, one of the highest in the world. These cameras are owned and operated by diverse stakeholders, including law enforcement, government bodies, private organizations, and individual citizens. However, despite this abundance of visual data, collaborative analysis remains a major challenge. City authorities and law enforcement agencies often require access to video footage from privately owned cameras for event reconstruction or investigation, yet private entities are reluctant to share raw footage due to privacy, trust, and data ownership concerns.

To address these challenges, we propose the Public Event Recording and Querying System (PERQS), a distributed and privacy-preserving framework for collaborative CCTV video analytics. PERQS integrates public, private, and individual cameras into a contributory network that enables secure querying of video feeds without requiring raw footage sharing. It employs local video analysis to maintain data sovereignty, while ensuring authenticity through hash-based video commitments that make tampering detectable. The system introduces two novel consensus mechanisms-query consensus, which ensures collective agreement across multiple cameras analyzing the same event, and time-travel consensus, which leverages spatiotemporal correlations from nearby cameras to fill blind spots or compensate for missing feeds. To facilitate flexible querying, we design a domain-specific query language, PERQL, inspired by SQL, which allows users to express analytical queries over distributed video data. The system's plug-and-play architecture supports seamless integration with diverse video analytics algorithms and computer vision models, enabling extensibility for future workloads.

To mitigate potential data leakage risks within the blockchain layer that underpins PERQS, we develop Aramid, a privilege separation mechanism for the Hyperledger Fabric permissioned blockchain platform. Aramid isolates peer nodes and enforces strict data access controls, ensuring channel confidentiality and reducing the attack surface within distributed deployments.

Recognizing that PERQS does not support joint video analytics natively, we further evaluate two leading privacy-preserving computation paradigms-Secure Multiparty Computation (MPC) and Trusted Execution Environments (TEE)-to enable collaborative analytics across organizations. We conduct a comprehensive experimental study covering five real-world case studies, including object re-identification, scene similarity detection, vehicle counting, and machine learning-based video fusion tasks. Additionally, we benchmark core image processing operations-such as thresholding, histogram computation, Sobel edge detection, convolution, and thinning-across various MPC configurations to determine the most efficient protocol for video workloads. Our results indicate that TEE-based approaches significantly outperform MPC, achieving up to 70-90× faster execution in compute-intensive tasks, while 3-party replicated secret-sharing (ring)-based MPC offers a reasonable trade-off in settings without trusted hardware. Through detailed evaluation of performance, security, and implementation complexity, we demonstrate that TEE and MPC represent complementary solutions for extending PERQS to support privacy-preserving, scalable, and trustworthy joint video analytics in multi-organizational environments.

Enabling Secure and Scalable Data Analytics on Edge Devices

A THESIS

SUBMITTED FOR THE DEGREE OF

Doctor of Philosophy

IN THE

Faculty of Engineering

BY

Arun Joseph

Computer Science and Automation
Indian Institute of Science
Bangalore – 560 012 (INDIA)

July, 2026

Declaration of Originality

I, **Arun Joseph**, with SR No. **04-04-00-10-12-18-2-16477** hereby declare that the material presented in the thesis titled

Enabling Secure and Scalable Data Analytics on Edge Devices

represents original work carried out by me in the **Department of Computer Science and Automation** at **Indian Institute of Science** during the years **2019-2025**.
With my signature, I certify that:

- I have not manipulated any of the data or results.
- I have not committed any plagiarism of intellectual property. I have clearly indicated and referenced the contributions of others.
- I have explicitly acknowledged all collaborative research and discussions.
- I have understood that any false claim will result in severe disciplinary action.
- I have understood that the work may be screened for any form of academic misconduct.

Date: Student Signature

In my capacity as supervisor of the above-mentioned work, I certify that the above statements are true to the best of my knowledge, and I have carried out due diligence to ensure the originality of the report.

Advisor Name: Advisor Signature

DEDICATED TO

My Family and Friends

who supported me this long journey

Abstract

Closed-Circuit Television (CCTV) systems play a vital role in public safety, urban monitoring, and criminal investigation by providing continuous visual surveillance. The proliferation of CCTV systems—exemplified by Delhi's 1,446 cameras per square mile—has created vast silos of visual data that remain inaccessible for collaborative public safety efforts due to privacy and ownership concerns.

To address these challenges, we propose the Public Event Recording and Querying System (PERQS), a distributed and privacy-preserving framework for collaborative CCTV video analytics. PERQS integrates public, private, and individual cameras into a contributory network that enables secure querying of video feeds without requiring raw footage sharing. It employs local video analysis to maintain data sovereignty, while ensuring authenticity through hash-based video commitments that make tampering detectable. The system introduces two novel consensus mechanisms-query consensus, which ensures collective agreement across multiple cameras analyzing the same event, and time-travel consensus, which leverages spatiotemporal correlations from nearby cameras to fill blind spots or compensate for missing feeds. To facilitate flexible querying, we design a domain-specific query language, PERQL, inspired by SQL, which allows users to express analytical queries over distributed video data. The system's plug-and-play architecture supports seamless integration with diverse video analytics algorithms and computer vision models, enabling extensibility for future workloads.

To mitigate potential data leakage risks within the blockchain layer that underpins PERQS, we develop Aramid, a privilege separation mechanism for the Hyperledger Fabric permissioned blockchain platform. Aramid isolates peer nodes and enforces strict data access controls, ensuring channel confidentiality and reducing the attack surface within distributed deployments.

Recognizing that PERQS does not support joint video analytics natively, we further evaluate two leading privacy-preserving computation paradigms-Secure Multiparty Computation (MPC) and Trusted Execution Environments (TEE)-to enable collaborative analytics across organizations. We conduct a comprehensive experimental study covering five real-world case studies, including object re-identification, scene similarity detection, vehicle counting, and ma-

Abstract

chine learning-based video fusion tasks. Additionally, we benchmark a suite of fundamental image-processing kernels across various MPC configurations to determine the most efficient protocol for video workloads. Our results indicate that TEE-based approaches significantly outperform MPC, achieving up to 70-90× faster execution in compute-intensive tasks, while 3-party replicated secret-sharing (ring)-based MPC offers a reasonable trade-off in settings without trusted hardware. Through detailed evaluation of performance, security, and implementation complexity, we demonstrate that TEE and MPC represent complementary solutions for extending PERQS to support privacy-preserving, scalable, and trustworthy joint video analytics in multi-organizational environments.

Publications based on this Thesis

- Arun Joseph, Nikita Yadav, Vinod Ganapathy, Dushyant Behl, and Praveen Jayachandran, *Data Protection in Permissioned Blockchains using Privilege Separation*, Proceedings of COMSNETS'23, the 15th International Conference on Communication Systems and Networks, Bangalore, India, January 2023.

- Arun Joseph, Nikita Yadav, Vinod Ganapathy, and Dushyant Behl. *A contributory public-event recording and querying system*, In Proceedings of the Eighth ACM/IEEE Symposium on Edge Computing, SEC '23, Wilmington, Delaware, USA, December 2023.

- Arun Joseph, Vinod Ganapathy, *SoK: Evaluation of Methods for Privacy Preserving Edge Video Analytics*, The 21st International Conference on Information Systems Security, ICISS'25, Indore, Madhya Pradesh, India, December 2025.

Contents

Abstract i

Publications based on this Thesis iii

Contents iv

List of Figures viii

List of Tables xi

1 Introduction 1

1.1 Motivation . . . 1
1.2 Problem Statement . . . 2
1.2.1 Scope and Contributions . . . 3
1.2.2 Assumptions and Limitations . . . 5
1.3 Design Overview . . . 6
1.4 Summary of Findings . . . 8
1.5 Thesis Organization . . . 11

2 Related Works 12

2.1 Video Querying Systems . . . 12
2.2 Video Analysis . . . 12
2.3 Privilege Separation . . . 13
2.3.1 Container Security. . . . 13
2.3.2 Smart Contract Security. . . . 14
2.3.3 Blockchain Security . . . 15
2.4 MPC Data Analytics . . . 15
2.5 TEE Data Analytics . . . 16

CONTENTS

3 A Contributory Public-Event Recording and Querying System 17
3.1 Introduction . . . 17
3.2 CCTV Systems and Video Recording . . . 19
3.2.1 Components of a CCTV System . . . 19
3.3 Example Use Cases . . . 20
3.4 Threat Model . . . 22
3.4.1 Participant and Data-Level Threats . . . 22
3.4.2 System Integrity Assumptions . . . 22
3.4.3 Privacy Considerations . . . 23
3.5 PERQS Design . . . 23
3.5.1 PERQS Architecture . . . 24
3.5.2 Public Event Recording and Committing . . . 25
3.5.3 Event Querying . . . 26
3.6 Query Execution Engine . . . 27
3.6.1 Query Language - PERQL . . . 27
3.6.2 Query Decoding and Execution . . . 29
3.6.3 Fuzzy Evaluation Procedure . . . 30
3.6.4 Video Analyzer . . . 31
3.6.5 Time-Travel Consensus . . . 32
3.6.5.1 Functional Query Adaptation . . . 32
3.6.5.2 Illustrative Example . . . 33
3.6.5.3 Core Principles . . . 34
3.6.5.4 Operational Procedure . . . 34
3.6.5.5 Limitations . . . 35
3.7 Experimental Setup . . . 36
3.7.1 PERQS Experiments . . . 36
3.8 Evaluation . . . 37
3.8.1 Benchmarking Framework . . . 37
3.8.2 Benchmark Evaluation . . . 39
3.8.3 Adversarial Experiments . . . 40
3.8.4 Accuracy and Performance . . . 42
3.8.5 Case Studies . . . 44
3.8.6 Baseline Discussions . . . 48
3.8.7 Optimizations . . . 50
3.8.8 Artifact Availability . . . 50

CONTENTS

3.9 Summary . . . 50

4 Aramid: Data Protection in Permissioned Blockchains 52

4.1 Introduction . . . 52
4.2 Hyperledger Fabric . . . 55
4.3 Threat Model for Aramid . . . 57
4.3.1 Comparison with Permissionless Blockchains . . . 58
4.4 Motivating Examples . . . 58
4.5 Aramid . . . 61
4.5.1 Enforcing Channel Data Separation . . . 62
4.5.2 Implementation . . . 65
4.6 Evaluation . . . 72
4.6.0.1 Setup for fabric testbed . . . 72
4.6.1 Evaluating Aramid's Robustness . . . 72
4.6.2 Performance Evaluation . . . 75
4.7 Summary . . . 77

5 Evaluation of Methods for Privacy Preserving Video Analytics 78

5.1 Introduction . . . 78
5.2 Secure Multiparty Computation (MPC) . . . 79
5.2.1 Cryptographic Foundations . . . 80
5.2.1.1 Secret Sharing . . . 80
5.2.1.2 Oblivious Transfer (OT) . . . 80
5.2.1.3 Garbled Circuits . . . 80
5.2.1.4 Homomorphic Encryption . . . 80
5.2.1.5 Zero-Knowledge Proofs (ZKPs) . . . 80
5.2.2 Fundamental Concepts . . . 81
5.3 Trusted Execution Environment (TEE) . . . 83
5.3.1 Fundamental Concepts and Properties . . . 84
5.4 Overview . . . 85
5.4.1 Case Studies . . . 86
5.4.2 Threat Model . . . 88
5.5 Experiment Setup . . . 88
5.5.1 Setup for MPC . . . 88
5.5.2 Setup for TEE . . . 90

5.6 Evaluation 92
5.6.1 Implementation Details 92
5.6.2 Choosing the Best MPC Protocol 93
5.6.3 MPC vs TEE Performance 95
5.6.4 Security Evaluation 96
5.6.5 Implementation Challenges 97
5.7 Summary 97

6 Conclusions and Future Directions **99**
6.1 Conclusion 99
6.2 Future Directions 101

Bibliography **102**

List of Figures

1.1 PERQS network and peer in edge servers of organizations to enable contributory video analytics. . . . 6
1.2 Extending PERQS with TEE/MPC for joint video analytics. . . . 8
1.3 Case Study 1: Majority Consensus - Four distinct blue trucks passing through the junction during the analyzed time interval. . . . 9

3.1 An example city map with CCTV cameras. Red cameras are the traffic police cameras, blue are the institution cameras, and green are homeowner's cameras. The map pointer represents the GPS location of these cameras. The intersections are marked on the road with alphabets. . . . 19
3.2 An IP camera CCTV system. Network Video Recorder stores the video and it is connected via router for remote access. . . . 20
3.3 PERQS components in video capturing organizations. PERQS peer enables participation in the PERQS network and handle the hash commitment and query execution. . . . 25
3.4 PERQL query execution stages. Organization 1 initiates a query execution. Sends the parsed query to two owner organizations, 2 and 3. . . . 29
3.5 Example 2 Section 3.3 illustration for time travel consensus. a) An accident occurred. b) Positions of the vehicles at $time = t - \Delta_2$. c) Positions at $time = t - \Delta_1$. d) Hit and run. Position at $time = t + \Delta_3$. Only the cameras which might capture the vehicles are shown. . . . 33
3.6 CityFlowV2 [49]: Case study 1: time sync screenshots of 5 cameras when a blue SUV truck crossing the junction at 11:30:05.50. . . . 36
3.7 Altered Camera 1 image, brightness reduced by 20% and contrast increased by 20%. . . . 40
3.8 Case study 2: cycle captured by camera 10, 11, 12, 13, 15 for the find cycle query at different times. . . . 46

LIST OF FIGURES

3.9 Case study 2: all the three cycles captured by camera 15 at 10:01:06.10. 47
3.10 Case Study 3: White truck captured by Cameras 16, 17, 18, 19, and 20 for the find white truck query. 49

4.1 Fabric network with single channel and two peers. 56
4.2 This figure compares how Fabric and Aramid organize peers. A single peer instance that is part of multiple channels in Fabric is replaced by multiple peer instances (one per channel) in Aramid. Each peer can run smart contracts (denoted by "S1", "S2", "S3") and can only access the ledger of that channel. A trusted proxy acts as a front for the multiple peer instances to external applications. The proxy inspects messages directed to the peers, and routes the messages appropriately, based on the channel information. In Aramid, peers are untrusted, and the trust is instead shifted to a proxy node. 63
4.3 Sizes of various components in Fabric (shaded gray) and Aramid. Peers and orderers are part of Fabric's TCB. The trusted proxy is part of Aramid's TCB, but peers and orderers are not. 64
4.4 Client/peer interaction in (a) Fabric; and (b) and Aramid. 67
4.5 Interaction between components in Aramid in fulfilling a transaction. In this example, a transaction initiated by the client executes a smart contract at Organization-1 and Organization-2, following which the endorsements are collected, and the endorsed transactions are sent to the ordering service. The orderer then broadcasts it to all the peers, who validate the transaction and add it to their ledgers. 69
4.6 Default configuration used for experiments. 73
4.7 Kubernetes cluster setup used in our experiments. The "Pods" column shows the main pods that we run in each cluster node. 73
4.8 Measuring the impact of transaction send rate on throughput and latency in both Fabric and Aramid. 75
4.9 Measuring the impact of the number of channels on throughput and latency in Fabric and Aramid. 76
4.10 Fabric and Aramid with six channels. 77

5.1 MPC protocol involves n parties, each holding private input x_i and computing a joint function $y = f(x_1, x_2, ..., x_n)$. 81

LIST OF FIGURES

5.2 Client sends encrypted sensitive data to the trusted TEE application. Other applications, operating systems and even hypervisor cannot access the sensitive data. 83
5.3 MPC outsourced model where secret shares of input is encrypted and send to the cloud servers. Then the servers will perform an MPC computation to evaluate the function on the secret input. This example is of a 3-party outsourced computation includes 3 independent servers. 90
5.4 An example of 3-party setup for MPC in-house model where participants execute protocol among themselves. Local computation and intermediate communications are required depending on the protocol used. 91
5.5 Three organizations perform collaborative data analytics using a trusted execution environment (TEE) available in a cloud server. 91

List of Tables

1.1 Number of CCTV cameras in some of the world's populated cities [7]. 2

3.1 Example Fuzzy Consensus Rules for Different Data Types. 30
3.2 Results of 40 randomly generated benchmark queries executed using three participating cameras. 38
3.3 Query execution performance across eight traffic junctions. Values denote the number of correct query responses generated by individual camera configurations (C1, C2, C3) and 3-camera (PERQS 3_Cams) and all-camera (PERQS ALL_Cams) consensus results. 40
3.4 Evaluation benchmark by varying Camera 1 to simulate different fault and adversarial conditions. The columns corresponding to Accuracy(A), Precision(P), Recall(R) and F1-Score(F1). 41
3.5 Case study 1: reply from five cameras available at junction one and the final result after majority consensus. 45
3.6 Case study 2: final result after time-travel consensus. 47

5.1 Details of the case studies include their critical computations, which are to be evaluated under privacy-preserving conditions, along with the available implementations and datasets. We use CityFlowV2 [49] for our analysis. 86
5.2 MPC protocols evaluated in this project. We use the MP-SPDZ [81] implementation. 89
5.3 MPC protocols performance varying sharing scheme and adversary type. We have measured execution time and global data sent for all protocols. Some results are unavailable because the MP-SPDZ [81] does not support it. The highlighted values are the best time and global data communication for 2-party and 3-party protocols. 93
5.4 MPC protocol performance when number of parties increases. 94

LIST OF TABLES

5.5 Performance of joint video analytics tasks of MPC and TEE. We used videos from CityFlowV2 [35] for all five workloads. Because of the lack of support and inherent memory limitations we are unable to run ml models using Intel SGX. . 95

Chapter 1

Introduction

1.1 Motivation

CCTV (Closed-Circuit Television) technology has become an invaluable asset for security and surveillance in various settings, including homes, businesses, public areas, and critical infrastructure. These systems consist of cameras, recorders, and monitors that enable real-time monitoring and recording of events. CCTV's significance stems from its ability to provide a visual record of events. It serves purposes such as deterring and monitoring criminal activity, furnishing evidence for criminal investigations, and enhancing public safety [115, 155, 169, 142, 6, 71, 9, 105]. CCTV use has increased in recent years due to technological advancements and increased demand for security and surveillance [8, 10].

The statistics in Table 1.1 reveal that Delhi, India, has 1446 cameras per square mile. These cameras are primarily installed by official traffic police and city authorities. In addition, many private organizations and individuals also have their own CCTV cameras that monitor public spaces. In London, there are over 942,562 CCTV cameras if we include private cameras [20], meaning there is one CCTV camera for every ten people in the city. A person is likely to be captured on London CCTV up to 70 times a day [20, 6]. Despite the abundance of CCTV cameras in neighborhoods, utilizing them collectively poses several complexities. The video streams captured by these cameras are owned by various entities, both private and public, making it difficult to query and access them as a unified resource.

In some situations, public authorities require access to additional footage captured by private cameras to aid their investigations. Let us consider a scenario where a car is involved in an accident and flees the scene. The identification of the vehicle becomes crucial in finding the responsible vehicle. Law enforcement agencies examine the traffic police CCTV footage, but if the captured view is limited, they explore additional sources, such as cameras installed by

City	# Cameras	Per Square Miles	Per 1000 People
Delhi, India	436600	1446	26.7
Chennai, India	282126	614	24.53
Singapore, Singapore	108981	281	18.04
Seoul, South Korea	77814	333	7.8
Moscow, Russia	213000	219	16.85
London, UK	127373	210	13.35
New York, USA	56190	187	6.87
Cities of China[1]	54000000	1000+	372+

Table 1.1: Number of CCTV cameras in some of the world's populated cities [7].

nearby businesses or residents. By analyzing multiple footage sources, authorities can cross-reference information to verify the registration number of the fleeing vehicle. This highlights the importance of a ***contributory CCTV network*** in assisting law enforcement in solving crimes and maintaining public safety.

Participating in contributory CCTV systems provides several benefits to organizations, both public and private. The main objective of CCTV is to enhance security and ensure the neighborhood's safety. By sharing public CCTV footage, the overall surveillance and security of the area can be significantly improved. In cases where one camera may be damaged, malfunctioning, or experiencing technical issues, other cameras from different organizations can still monitor the area, ensuring continuous surveillance coverage. These observations underline the need for a unified yet privacy-preserving framework that can integrate disparate CCTV sources without compromising ownership or trust.

1.2 Problem Statement

Building a comprehensive contributory CCTV network system poses several challenges. Private entities are often reluctant to share their video feeds with public authorities. Concerns regarding privacy, data security, and trust in the appropriate use of footage contribute to this hesitation. Additionally, the reliability, camera quality variations, time synchronization, and blind spots in surveillance coverage make it difficult to create an effective system. The four primary challenges in designing such a system are as follows:

① **Untrustworthy Video Feeds:** Video feeds are compromised by technical errors or malicious activities, resulting in damaged or edited footage that is unreliable for investigations.

[1]Due to limited transparency, the exact number of cameras in each Chinese city is still being determined. However, if China has 540 million cameras and divides it among its 1.46 billion population, we can estimate that there are approximately 373 cameras per 1,000 people.

Furthermore, a lack of trust exists among organizations, making video footage sharing difficult. There is always the risk that organizations might intentionally share incorrect footage to benefit themselves or to deceive others.

② **Privacy of Video Feeds:** Private organizations do not wish to share raw video feeds. They are concerned about the security and privacy of data, as well as the potential for unauthorized access and misuse of the footage. These concerns make it difficult to get all parties to participate in a contributory CCTV network.

③ **Variable Quality of Video Feeds:** The cameras used for recordings differ in quality, viewing angle, aspect ratio, and color calibration, making it difficult to standardize and integrate the footage for analysis. Additionally, the recording timestamps of different cameras may not be perfectly synchronized, leading to inconsistencies in the data. Moreover, the sheer volume of data generated by multiple cameras will make it challenging to process and analyze the footage efficiently.

④ **Availability of CCTV Footage:** The absence of CCTV cameras in certain areas results in blind spots, leaving surveillance systems incomplete. In addition, camera malfunctions also lead to gaps in footage. Even without direct visual evidence, it is possible to use nearby cameras to infer what happened. However, identifying the appropriate neighboring cameras and effectively utilizing them is a challenging task.

While edge devices offer new opportunities for real-time and context-aware data analytics, they also introduce significant challenges. Performing analytics on distributed, resource-constrained devices raises concerns about data confidentiality, integrity, and performance. There is a critical need for a comprehensive framework that enables secure and scalable data analytics at the edge, integrates privacy-preserving techniques, and can function reliably in collaborative multi-party settings such as public surveillance systems or inter-organizational environments.

1.2.1 Scope and Contributions

We propose the Public Event Recording and Querying System (PERQS), a blockchain-based framework for secure, consensus-driven video analytics to address these challenges.

PERQS fundamentally changes how collaborative video analysis is performed. Instead of requiring users to manually inspect feeds or trust specific camera owners, PERQS allows users to submit high-level queries. The system then automatically coordinates analysis across participating cameras and returns verified results. This design removes the need for blind trust while ensuring that responses are collectively validated. By embedding consensus into the video an-

alytics workflow, PERQS directly addresses the core problems of trust, privacy, and reliability in contributory surveillance environments.

To achieve this, PERQS introduces several key mechanisms, each motivated by a specific limitation in existing systems. First, video-hash commitments ensure tamper-resistance recording of footage metadata, preventing post-event manipulation or disputes about authenticity. Second, privacy-preserving result sharing allows only analysis outputs—not raw video—to be exchanged, eliminating centralized storage risks and protecting sensitive visual data. Third, query consensus guarantees robust results even if some cameras malfunction or act maliciously. Finally, a novel time-travel consensus mechanism enables the system to infer missing observations using spatially and temporally adjacent cameras, thereby mitigating blind spots and incomplete coverage.

Together, these mechanisms establish a secure and verifiable foundation for collaborative video analytics.

The requirements for trust minimization, verifiable commitments, and distributed consensus naturally motivate a blockchain-based architecture. We implement PERQS using the permissioned platform **Hyperledger Fabric**, which supports distributed ledgers and smart contracts across organizational boundaries. However, while Fabric provides modularity and private data collections, its containerized peer architecture can still expose confidential data to insider threats or compromised components.

To strengthen confidentiality guarantees, we design Aramid, an enhanced version of Fabric that partitions peers per channel and replaces complex trusted components with smaller, auditable proxy modules. This redesign minimizes the trusted computing base and significantly reduces the attack surface, thereby improving practical data protection in permissioned blockchain deployments.

PERQS currently limits direct joint video processing across organizations. To address this limitation, we systematically evaluate two major approaches for secure cross-party computation: Secure Multiparty Computation (MPC) and Trusted Execution Environments (TEE). We analyze these approaches not only from a theoretical security perspective, but also from a systems viewpoint. Specifically, we evaluate:

- Computational efficiency, since video analytics tasks such as object detection and re-identification are resource-intensive.
- Communication overhead, which is critical for distributed edge deployments.
- Scalability, given that real-world deployments may involve hundreds of cameras.
- Security guarantees, comparing cryptographic assurances of MPC with hardware-based iso-

lation in TEEs.

- Deployment practicality, including integration complexity and operational constraints.

This comparative study provides clear guidance on the trade-offs between cryptographic and hardware-assisted approaches for secure edge video analytics.

Scope of the Thesis. This research focuses on secure data analytics frameworks for edge environments, with emphasis on: (i) privacy-preserving collaborative analytics, and (ii) contributory video surveillance systems with verifiable guarantees.

Main Contributions. The main contributions of this thesis are:

- **PERQS:** A blockchain-based Public Event Recording and Querying System that enables trustless, privacy-preserving collaborative video analytics using consensus and tamper-resistance commitments.
- **PERQL:** A domain-specific query language that enables expressive, scalable, and structured querying of distributed video feeds.
- **Aramid:** A strengthened confidentiality architecture for permissioned blockchains that reduces the trusted computing base and mitigates insider and container-level threats.
- **Secure computation evaluation:** A comprehensive systems-level comparison of MPC and TEE approaches for practical, scalable, and secure multi-party video analytics.

Prior works related to video analytics and querying are not equipped to solve many of these challenges. [77, 17, 16, 180, 123, 78, 171] can be used to query large set of videos with the help of video analytic tools. They assume videos are readily available for analysis. None of them addresses video privacy, correctness, or mutual trust between participants. While [31, 121, 163, 164, 175] attempt to address the privacy issues to an extent. A detailed related works discussion can be found in Chapter 2.

Unlike prior works that rely on centralized access or mutual trust among participants, PERQS introduces a decentralized, privacy-preserving framework for collaborative CCTV analytics that guarantees data authenticity through blockchain commitments, enforces reliability through query consensus, and preserves privacy by sharing only analytical outcomes rather than raw footage.

1.2.2 Assumptions and Limitations

PERQS adopts a zero-trust assumption at the participant level, where camera owners, edge nodes, and participating organizations are not assumed to trust one another. The system

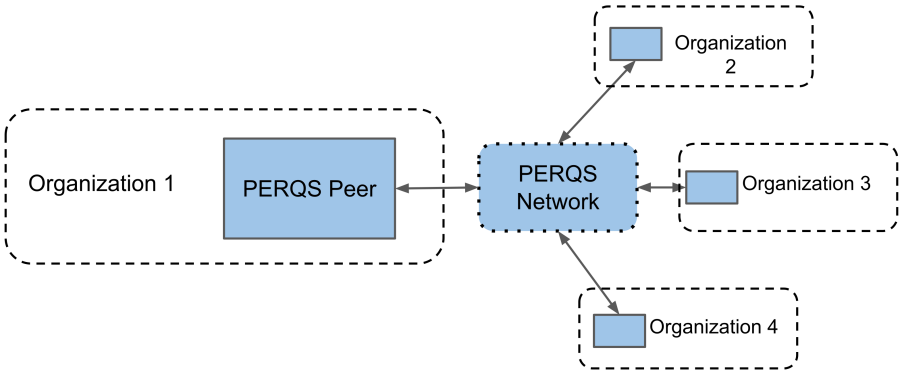

Figure 1.1: PERQS network and peer in edge servers of organizations to enable contributory video analytics.

is designed to detect inconsistencies and provide verifiable analytics through consensus and tamper-resistance commitments. However, while PERQS guarantees the integrity and traceability of video data after registration into the system, it does not prevent manipulation at the point of capture. If a camera is physically compromised or intentionally records falsified content, such capture-time attacks remain outside the protection scope of the framework.

The effectiveness of the proposed consensus mechanisms depends on the availability of sufficient camera coverage and the quality of the underlying video analytics models. Majority consensus requires overlapping observations from multiple cameras, while time-travel consensus depends on the availability of nearby cameras capable of capturing prior or subsequent appearances of the target object. In addition, the correctness of query results is inherently bounded by the accuracy of the deployed machine learning models, such as object detection and re-identification systems. Errors in these models may propagate into the consensus process and affect the final query outcome.

Aramid shifts trust away from application-layer blockchain components and toward a minimal infrastructure-level enforcement layer in order to reduce the Trusted Computing Base (TCB). Instead of trusting the full blockchain peer implementation and associated business applications, Aramid relies on infrastructure isolation mechanisms provided by container orchestration and service-mesh frameworks such as Kubernetes and Istio.

1.3 Design Overview

PERQS is designed to connect public, private, and individual CCTV cameras into a contributory distributed network that enables collaborative video analytics. We adopted a distributed design to respect the privacy of participating organizations and individuals, many of whom may

wish to keep their raw video feeds private. To achieve this, PERQS introduces a distributed storage and analytics architecture that eliminates the need to share raw footage, allowing only processed results to be exchanged across participants.

A key objective in our design was to ensure system reliability, even when some participants behave unpredictably due to technical faults or malicious intent. To meet this goal, we adopted a blockchain-based architecture that inherently supports trustless collaboration and data integrity. In this design, each participant commits a hash of the video feed to the blockchain at the time of recording, enabling later verification of video authenticity and detecting any tampering or corruption. Furthermore, query execution and consensus mechanisms are seamlessly integrated into the blockchain, allowing distributed query processing without requiring access to raw video data. Figure 1.1 illustrates the PERQS network and blockchain peers that enable participant integration.

We chose to use a permissioned blockchain model, as the intended participants—CCTV owners, government agencies, and other authorized authorities—are known and authenticated entities. This approach allows for fine-grained access control, with defined roles and permissions, while maintaining transparency and accountability among known entities. Additionally, the blockchain's distributed nature enhances fault tolerance, ensuring that the system remains operational even if some nodes fail. Overall, the design of PERQS provides an efficient, scalable, and privacy-preserving platform for querying and analyzing video footage across large, diverse CCTV networks. Chapter 3 describes the PERQS architecture and query execution mechanisms in greater detail.

Although PERQS provides a robust framework for video querying, its reliance on permissioned blockchains like Hyperledger Fabric introduces a new attack surface: the potential for data leakage across channels. This motivates the design of Aramid in Chapter 4.

While PERQS efficiently supports distributed and privacy-preserving video analytics, it does not natively enable joint video analytics, where multiple organizations' video feeds must be processed together to extract insights spanning across cameras or locations. To address this limitation, we propose integrating TEE or MPC as complementary computation layers. As illustrated in Figure 1.2, each participating organization retains its local video storage and performs initial analytics at its own edge server, maintaining data locality and privacy. When joint analysis is required, only the relevant encrypted features or intermediate representations are transmitted through secure channels to a trusted computation layer implemented using TEE or MPC.

In this architecture, the TEE-based approach executes the joint analytics inside a secure hardware enclave on the cloud, ensuring that even the cloud operator cannot access the raw

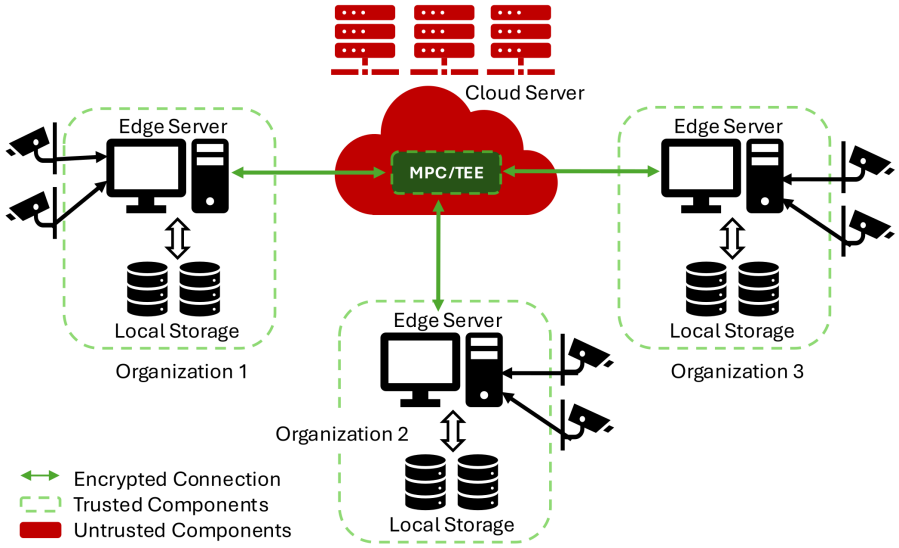

Figure 1.2: Extending PERQS with TEE/MPC for joint video analytics.

data. Alternatively, the MPC-based approach distributes computation across multiple parties, ensuring that no single entity learns another's private data during the computation. Both methods remove the need to place blind trust in the cloud service provider, instead relying on cryptographic or hardware-based guarantees of confidentiality. The design preserves privacy while enabling collaborative analytics—such as object tracking across cameras owned by different organizations—without exposing sensitive video data. Thus, combining PERQS with TEE or MPC significantly extends its capabilities, allowing secure, cross-domain video analytics while maintaining the principles of trust minimization, scalability, and data sovereignty that underpin the original PERQS design. The relative merits and demerits are evaluated in detail on Chapter 5.

1.4 Summary of Findings

For the evaluation of PERQS, we developed a benchmark framework using the CityFlowV2 dataset and conducted a series of case study–based experiments to validate the effectiveness, robustness, and scalability of the proposed consensus mechanisms under realistic traffic surveillance conditions. The benchmark consisted of 40 automatically generated vehicle queries executed across eight traffic junctions with varying camera coverage and observation quality. The evaluation measured Accuracy, Precision, Recall, and F1-score under normal, faulty, and adversarial operating conditions.

Majority Consensus. In case study 1, multiple CCTV cameras monitoring the same traffic junction analyzed overlapping video streams. The objective was to determine whether PERQS

Figure 1.3: Case Study 1: Majority Consensus - Four distinct blue trucks passing through the junction during the analyzed time interval.

could reliably detect events despite inconsistent detections, viewpoint variations, degraded video quality, and faulty inputs. As illustrated in Figure 1.3, four trucks passed through the monitored junction. Using majority consensus, PERQS correctly detected all four vehicles through collaborative reconciliation across cameras. Across the benchmark queries, consensus-based aggregation significantly improved reliability compared to isolated camera feeds.

Time-Travel Consensus. We then evaluated the system under partial camera failures and blind-spot conditions. Five of the eight evaluated junctions lacked fully overlapping camera coverage, requiring PERQS to reconstruct events using neighboring cameras and temporal offsets. Selected camera feeds were intentionally omitted to simulate outages and incomplete visibility. PERQS leveraged spatiotemporal correlation and trajectory reconstruction to infer vehicle appearances across nearby routes. Even when direct footage from a queried location was unavailable, the system successfully recovered events through neighboring observations, significantly improving Recall compared to isolated cameras. This demonstrates that PERQS can maintain reliable query execution even in partially connected surveillance deployments where direct visibility is unavailable.

We additionally evaluated PERQS under adversarial and degraded operating conditions by simulating faulty cameras, poor detector configurations, and Byzantine participants reporting incorrect results. The experiments show that while individual camera accuracy degraded significantly under noisy or malicious conditions, collaborative consensus maintained substantially higher Precision and F1-score by filtering inconsistent detections through collective agreement. Even when one camera continuously produced incorrect outputs, PERQS maintained reliable query results as long as an honest majority of participants existed.

Finally, we performed detailed performance profiling of the complete query execution pipeline,

including query planning, distributed video analytics, result aggregation, and consensus verification. The results show that machine learning inference dominates the overall execution cost, while consensus formation introduces comparatively small overheads. Detailed quantitative metrics are provided in Chapter 3.

In addition to the PERQS evaluation, we demonstrated the privilege separation architecture built on Hyperledger Fabric to mitigate the risk of channel data leakage in Chapter 4. Traditional permissioned blockchain deployments often face measurable confidentiality risks when multiple channels coexist on shared infrastructure. To address this, we implemented a modified Fabric architecture that separates peer nodes and associated components on a per-channel basis, thereby strictly isolating ledger data across channels.

To evaluate the performance impact of this isolation, we conducted experiments by varying the number of peers, channels, and transaction loads, and measured system throughput and latency under increasing scale. The results show that privilege separation introduces additional overhead due to process isolation and channel-level partitioning. However, throughput degradation remains bounded and predictable, and latency increases remain within acceptable operational limits. These findings demonstrate that the confidentiality gains achieved through channel isolation justify the moderate performance trade-off, especially for sensitive multi-organization deployments.

In Chapter 5, we conducted a detailed benchmarking study comparing Secure Multiparty Computation (MPC) and Trusted Execution Environments (TEE) for joint video analytics. Across fundamental image-processing primitives (thresholding, histogram computation, Sobel edge detection, convolution, and thinning), the three-party replicated secret-sharing (ring) MPC model consistently achieved the best performance among evaluated MPC protocols, providing the most favorable balance between security guarantees and execution time.

MPC vs. TEE Performance Gap. Using five representative video analytics case studies, we measured runtime overheads under realistic workloads. The results show:

- For lightweight analytics tasks, MPC was approximately 3–4× slower than TEE.
- For computation-intensive tasks (e.g., deep neural network inference or multi-frame aggregation), MPC incurred a 70–90× slowdown compared to TEE.
- Communication overhead significantly contributed to MPC latency, especially under large intermediate tensor exchanges.
- TEE-based execution maintained near-native performance with minimal communication overhead.

These measurements establish the performance boundary between the two paradigms: TEE

offers near-native execution efficiency suitable for real-time analytics, whereas MPC provides stronger decentralization guarantees at substantial computational cost for heavy workloads. The evaluation provides concrete performance thresholds that guide system designers in selecting the appropriate secure computation paradigm depending on workload intensity and trust assumptions.

Beyond technical validation, the measured results show that privacy, trustlessness, and accountability can be achieved without rendering distributed surveillance systems impractical. This work therefore provides both architectural innovations and empirically grounded performance insights for deploying secure multi-party video analytics in real-world edge environments.

1.5 Thesis Organization

The remainder of this thesis is structured as follows: Chapter 2 surveys related works in secure video querying, edge analytics, and privacy-preserving computation. Chapter 3 introduces the design and implementation of PERQS, along with its query execution engine and consensus mechanisms. Chapter 4 demonstrated the privilege separation architecture to enhance privacy and security of participants. Chapter 5 presents an empirical evaluation of MPC and TEE for edge data analytics using real-world workloads. Chapter 6 concludes the thesis and outlines future research directions.

Chapter 2

Related Works

2.1 Video Querying Systems

Numerous studies on video querying systems have focused on various aspects, including feature extraction, query interface design, and query optimization. For instance, Blazeit [77] proposed a system for efficiently performing spatiotemporal queries over large video datasets. They introduced FRAMEQL, a declarative extension of SQL for video analytics, which enabled optimized aggregation and cardinality-limited queries, resulting in up to 83x performance improvement. While [75] is a user-friendly end-to-end video query system.

In contrast, the QBIC system [50] was developed by IBM researchers in the 1990s for searching and retrieving images from large collections of visual data based on their content. Another study [17] discusses the implementation and evaluation of a video query processing system in the VDBMS Testbed, while [180], [16] focuses on the challenges of performing real-time video analytics on large-scale live video streams. Scanner [123] is a distributed system that enables users to perform various video analytics tasks on large-scale video data, such as object detection, tracking, and scene understanding. Miris [22] is a system designed to query object tracks in videos efficiently.

Additionally, VIVA [78] is an end-to-end system that allows users to interact with video data using natural language queries. At the same time, PRIVID [31] is a practical privacy-preserving framework for video analytics queries. Finally, Vstore [171] is a data store designed to perform analytics on large videos.

2.2 Video Analysis

Several deep learning architectures have been developed for object detection, video classification, and action recognition in videos. YOLOv3 [128] uses a powerful backbone network

based on Darknet-53 for real-time object detection, while Faster R-CNN [131] significantly improves the speed of object detection. 3D Convolutional Networks (3D CNNs) [149] process video frames as a 3D stacked frame volume, including temporal information for video analysis. Non-local Neural Networks [167] capture long-range dependencies in images and videos. Convolutional Neural Networks (CNNs) are used in [79] for video classification, while [157] improves video classification accuracy using self-supervised learning for 3D CNNs. Anomaly detection in videos is tackled in [55] using self-supervised and multi-task learning. Two-stream Convolutional Networks [139], involving two separate CNNs, are used for action recognition in videos, with TS-BiRCNN [140] improving this architecture. [165] aims to address the challenge of modeling long-term temporal dependencies in videos while maintaining computational efficiency.

In addition to the mentioned systems, Visor [121] provides confidentiality for both the user's video stream and the machine learning models in the event of a compromised cloud platform. Vigil [183] is a real-time distributed wireless surveillance system that leverages edge computing capabilities. Chameleon [72] is a controller that dynamically selects optimal configurations for existing neural network-based video analytics pipelines. EdgeEye [100] is an edge computing framework specifically tailored for real-time intelligent video analytics applications. Reducto [93] is a system that adaptively adjusts filtering decisions based on time-varying correlations in the video data. CrossRoI [60] utilizes the inherent physical correlations of cross-camera viewing fields to enhance video analytics performance. Lastly, Spatula [70] enables scalable cross-camera analytics by leveraging edge compute boxes.

2.3 Privilege Separation

There is a long history of work on privilege separation in the systems security community. Privtrans [30] is an early tool that aimed to automatically transform source code into multiple, privilege-separated components, using programmer-supplied annotations. Privilege separation methods have also been investigated for specific kinds of applications, such as databases [48], HTML5 Web applications [11, 12], Web servers [87] and Android applications [120]. Diesel [48], which applies privilege separation for database access.

2.3.1 Container Security.

Several works have analyzed the security of container implementations, and many vulnerabilities have been found [158, 161, 159, 119, 160]. An attacker could exploit these vulnerabilities to write to arbitrary files, execute arbitrary code (with root privileges), set arbitrary Linux Security Modules (LSM), or gain access to control regions of the device attached to the host PCI bus.

Lin *et al.* [94] collect an attack dataset of 223 exploits resulting from 148 vulnerabilities. They categorize the attacks based on the impact of the attack and the location of the vulnerable component. They find that 56.8% of these exploits are effective against default container configurations, and they also propose a defense mechanism to defeat privilege escalation attacks.

Many of the exploits arise from vulnerabilities hidden in the container images. Shu *et al.* [138] utilize ClairOS [37] to analyze 356,218 docker images. They find that, on average, an image contains more than 180 vulnerabilities, which commonly propagate from the parent image to the child image. Tak *et al.* [146] highlight another important cause of security exploits on container platforms, *viz.*, updates to live public containers. Gao *et al.* [52] investigate in-container information leakage channels that expose host and co-resident containers' information. They show that using such information, an attacker can mount an advanced power attack.

Side-channel attacks [173, 68] also apply to containers, as shown by Zhang *et al.* [184]. They demonstrate cross tenant side-channel attack on PaaS cloud where tenants are isolated using Linux containers. Several researchers [85, 19, 109] employ container migration as a defense solution to co-resident attacks on the cloud. Migrate [19] minimizes the probability of attacker-victim co-residency on the same host, achieved by probabilistic random migrations of cloud tenants' containers between different hosts. Several works propose design changes to container implementations to enhance their security. The proposed methods include, for example, (a) employing proxies to mediate all commands from end-users to the docker daemon for least-privilege enforcement [182]; (b) using lightweight VMs as a faster and safer replacement for containers [103]; and (c) improved security policy enforcement using namespaces [144] and using `seccomp` [56]. Bastion [111] takes the approach of redesigning a network stack per container to construct an inherently secure container networking system. It prevents broad classes of network attacks, including L2/3/4 attacks, traffic eavesdropping, host service access, and host network namespace abuse. Aramid can also further benefit from all these proposed container-hardening defenses, but takes the orthogonal approach of attack containment and data isolation in case a container does get compromised.

2.3.2 Smart Contract Security.

Bugs in and exploits against smart contracts have led to huge financial losses in the past. In the DAO attack [46], the attacker stole more than $50 million by exploiting a reentrancy vulnerability. As another example, over $30 million was reported lost in the Parity Multi-sig Wallet exploit [185]. Such attacks have raised serious concerns regarding smart contract security. Several static analysis based tools have been proposed to detect bugs in smart contracts. These tools can detect bugs that include transaction-ordering dependence, mishandled

exceptions, reentrancy [102], logically incorrect or unfair contracts, bugs due to the miner's influence [76], and bugs that manifest from multiple invocations of a smart contract [114]. In addition to bug-finding tools, there have also been efforts to formally verify smart contracts for functional correctness (primarily in the Ethereum blockchain, for smart contracts using Solidity [141]) using state reachability-based methods [133], bounded model checking [51], and relational reasoning [28, 181].

2.3.3 Blockchain Security

Confidentiality of data for blockchain applications has in the past been supported using cryptographic mechanisms in the application layer (*e.g.,* [25, 112, 34, 122]). Zerocash [25], for instance, extends the Bitcoin protocol by adding new types of transactions on a separate privacy-preserving currency, where the transactions hide the sender, the recipient as well as the quantum of payment. This is achieved using zero knowledge proofs, specifically zk-SNARKs. ZKLedger [112] leverages Schnorr-type non-interactive zero-knowledge proofs to ensure payments are only known to the participants of the transaction but are publicly verifiable and auditable, *i.e.,* the transactions cannot be hidden from an auditor.

Recent work has leveraged secure multiparty computation (MPC) to support general-purpose computation on private data, leveraging blockchain for properties such as auditability, verifiability and fairness [189, 26, 186, 127]. Enigma [189] provides a peer-to-peer network platform for performing computations jointly on data while keeping the data entirely private. It provides an optimized version of MPC, with blockchain providing access control, a tamper-proof log of events (a bulletin board for MPC computations) and an incentive mechanism to ensure fairness. Benhamouda *et al.* [26] propose an extension to Hyperledger Fabric where the peers store on encrypted private data on the ledger, and use secure MPC whenever such private data is required in a transacion. This enables transactions to combine public and private data. Other work leveraging MPC with blockchain is covered in [127, 186].

2.4 MPC Data Analytics

Multiparty Computation offers a way to perform analytics without compromising data privacy. [113] demonstrated how MPC can be used for secure statistical analysis, such as regression analysis, by distributing the computation among multiple parties to ensure data privacy. [137] introduced a framework for privacy preserving deep learning using MPC, which allows multiple parties to collaboratively train a neural network without revealing their datasets. [108] proposed SecureML, a system that enables privacy-preserving machine learning inference using MPC. Their work demonstrates that performing secure prediction with minimal overhead is feasible.

[150] presented a federated learning approach that leverages MPC to ensure the confidentiality of local model updates during the aggregation process. [107] is a mixed protocol framework designed to enable efficient and privacy-preserving machine learning, combining arithmetic and Boolean secret sharing techniques. [41] introduced techniques to reduce the communication complexity of MPC protocols, making them more practical for large-scale applications.

2.5 TEE Data Analytics

Trusted Execution Environments provide a secure area within a processor that ensures the confidentiality and integrity of data and code. [38] explored the use of Intel SGX for secure data aggregation, enabling analytics on encrypted data without exposing raw data to the aggregator. [116] demonstrated using TEE to perform privacy-preserving machine learning, allowing models to be trained on sensitive data without compromising privacy. [64] presented techniques for processing encrypted video data within TEE without exposing the content to the host system.

No extensive studies have compared MPC frameworks and TEE frameworks specifically for shared video analytics workloads. Although both MPC and TEE offer robust methods for ensuring data privacy and security, their comparative effectiveness and performance in video analytics remain underexplored.

Chapter 3

A Contributory Public-Event Recording and Querying System

3.1 Introduction

In this chapter, we introduce the ***Public Event Recording and Querying System (PERQS)***. Our system aims to establish an efficient distributed network of CCTV cameras, overcoming the need for participants to place blind trust in one another. The core idea behind PERQS is to create a collaborative environment where users submit queries to the system, and it will autonomously utilize the entire network of camera feeds to provide responses by leveraging advanced video analytic techniques. This eliminates the need for users to review individual camera feeds manually or rely on the cooperation of specific participants within the network. PERQS introduces a novel consensus based video analysis concept to solve trust and privacy challenges associated with a contributory CCTV camera network. The insights we put together to solve the main challenges are as follows.

① **Tamper-resistant Video feed:** PERQS introduces video-hash commitment and verification, which assures video authenticity. In case of disputes, the authenticity of the video footage can be easily verified, eliminating any concerns about the video being edited or tampered with later on. The primary goal of this mechanism is to provide verifiable proof of data integrity for stored footage.

② **Private Video Storage:** PERQS does not share the raw video feeds. Instead, the system only shares the video analysis results, ensuring that sensitive information is kept private. This approach enhances privacy and eliminates the need for a vast central storage facility to store massive amounts of video data.

③ **Consensus-based Video Analysis:** To address the reliability issues of individual cameras, PERQS uses consensus, where multiple cameras participate in a query execution to reach a result. This approach ensures that the camera's fault, damage, or malicious behavior does not affect the output. Consensus not only improves the accuracy of the surveillance but also reduces the likelihood of false positives.

④ **Time-travel Consensus:** If there are blind spots in the city or camera faults, we devised a novel time-travel consensus mechanism in PERQS that uses footage from other cameras to answer the query. This mechanism can also be used when there are insufficient cameras to reach a consensus. The system uses nearby cameras to access pre-event or post-event footage to achieve consensus. This process is known as time-travel consensus because it enables the system to "travel through time" and use footage from other cameras to fill in gaps in the recordings.

The requirements of mutual distrust between participants, the hash-based commitment of video feeds, and the consensus naturally led us to adopt a blockchain-based architecture. This also helps us to ensure that participants do not need to trust each other for either the video feeds or the analysis results. Consensus and hash verification enable us to establish collective correctness, ensuring the accuracy and reliability of the system. To enable a flexible and customizable user experience, we developed the PERQL query language, similar to SQL, an extension of FRAMEQL[77]. This language allows users to design queries tailored to their specific needs while also providing the ability to combine multiple queries to answer more complex questions. Each participant maintains their analysis facility for the video analysis, and our architecture ensures that any video analysis tool can be easily plugged into the system. We combine the novel query consensus, time-travel consensus, tamper-resistant videos, and query language in PERQS to address all of the challenges previously discussed.

The novel contributions in this chapter are:

- Design of a distributed collaborative CCTV video querying framework, **PERQS**, its prototype implementation and evaluation.
- ***Query consensus*** mechanism to solve mutual trust and video quality-related issues.
- ***Time-travel consensus*** to improve the query consensus with the help of additional footage from nearby cameras.

Figure 3.1: An example city map with CCTV cameras. Red cameras are the traffic police cameras, blue are the institution cameras, and green are homeowner's cameras. The map pointer represents the GPS location of these cameras. The intersections are marked on the road with alphabets.

3.2 CCTV Systems and Video Recording

Closed-Circuit Television (CCTV) systems are surveillance infrastructures designed to capture, transmit, and store video footage for security, monitoring, and analytical purposes. Unlike broadcast television, CCTV operates on a closed communication loop, meaning the video signals are not publicly transmitted but are accessible only to authorized users. Over the years, CCTV technology has evolved from analog tape-based systems to sophisticated, networked video surveillance architectures capable of real-time monitoring, intelligent analytics, and remote access.

CCTV systems form the backbone of modern public safety, traffic management, industrial automation, and smart city infrastructures. They generate continuous visual data streams that can be analyzed to detect incidents, recognize patterns, and support decision-making processes in law enforcement and urban management.

3.2.1 Components of a CCTV System

- **Cameras (Image Acquisition Devices):** Used to capture live video streams from the environment. Modern IP cameras integrate local processing (edge analytics), enabling object detection, motion tracking, and event triggers.

- **Transmission Medium:** Transfers video data from cameras to recording or display units.

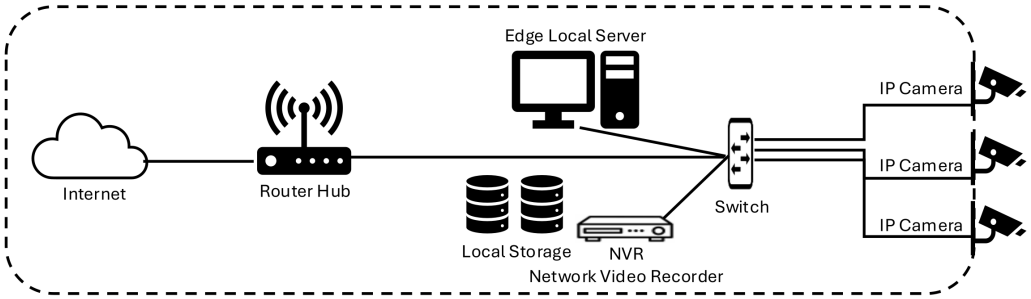

Figure 3.2: An IP camera CCTV system. Network Video Recorder stores the video and it is connected via router for remote access.

- **Recording and Storage Units:** Responsible for capturing and archiving video streams. DVR (Digital Video Recorder) used in analog systems; converts video to digital form and stores locally. NVR (Network Video Recorder) used in IP systems; receives digital streams directly from network cameras. Cloud Video Recorders (CVR) store encrypted video data in cloud environments, offering scalability and remote access.

- **Monitoring Interface:** Enables live or playback viewing through control rooms, mobile devices, or web interfaces.

- **Control and Management Software:** Manages camera configuration, video retrieval, and user permissions. Advanced systems support Video Management Systems (VMS), event-based triggers, and integrated analytics modules.

3.3 Example Use Cases

A contributory network of CCTV cameras can significantly improve large-scale video monitoring and event analysis. By collaboratively utilizing observations from multiple cameras, the system can generate more reliable and accurate results than isolated camera deployments. We use the map in Figure 3.1 to illustrate several practical scenarios where collaborative querying provides clear advantages.

Example 1: Hit and Run Incident. A car involved in an accident flees the scene. The objective is to locate the culprit by identifying the car involved. Consider the map in Figure 3.1. Suppose the accident happened at junction *'D'*, and the traffic police CCTV nearby only captured a side view of the vehicle. According to the figure, the car originated from somewhere between *'B'* and *'D'* and proceeded towards *'E'*. Unfortunately, there are no other traffic police cameras available to identify the license plate number. However, there is a camera at junction

'C' and a few individual cameras on the way towards *'A'*. By utilizing this supplementary footage, we can pinpoint the individuals responsible for the accident.

Example 2: Incident in a Camera Blind Area. Consider a scenario where two vehicles collide in an area without CCTV cameras. Suppose that one vehicle's excessive speed causes an accident. Video footage can be gathered from homeowners, shop owners, and ATMs along the road to estimate the car's speed. Combined, these pieces of evidence can support the claim of an overspeeding accident. Say a truck travelled from *'B'* to *'M'*, a car from *'M'* to *'C'* and the accident occurred between *'D'* and *'I'*. There are a couple of traffic police cameras along the road and many other private cameras. They can all participate in the over-speed estimation process.

Example 3: Vehicle counting. Counting the number of cars on a particular road is a critical task for traffic management and infrastructure planning. This information helps authorities assess traffic conditions, identify congested areas, and make informed decisions about road widening or constructing additional routes to alleviate traffic congestion. However, challenges arise when the cameras are positioned on the side of the road, as they may not capture all passing vehicles. For example, if a camera is focused on one side of the road, it may not detect cars traveling alongside large vehicles, such as buses or trucks. This limitation can lead to undercounting and inaccurate representation of the traffic volume. Leveraging multiple cameras in this scenario can provide a higher level of accuracy and confidence in the car count.

Example 4: Road Safety Violation. Traffic safety is a vital concern in every city and town around the world. In order to keep roads safe for drivers, passengers, pedestrians, and cyclists, it is essential to enforce traffic rules and regulations. Accidents can occur when vehicles fail to adhere to traffic safety regulations, such as running red lights, speeding, or riding a bike without wearing a helmet. It is crucial to identify these violations and issue fines or warnings to those responsible. While traffic police cameras may not provide comprehensive coverage of all areas within the city, by collaborating with other stakeholders in the community, traffic police can access additional footage from these privately owned cameras to help identify and apprehend traffic violators more accurately. This additional information can be used to investigate accidents, determine who is at fault, and enforce traffic laws. Ultimately, a collaborative approach to surveillance can lead to safer streets, fewer accidents, and a better quality of life for everyone in the community.

As we have observed, the pooling of CCTV footage from a variety of sources has the potential to enhance public safety greatly. However, there are also several challenges that come with bringing together footage from private, public, and individual organizations. We present PERQS (Public Event Recording and Querying System) as a solution to the challenges of

pooling CCTV footage from various organizations.

3.4 Threat Model

PERQS adopts a **zero-trust assumption** at the participant level. Although the system operates in a permissioned environment where organizations are authenticated, their software stacks, infrastructure, and operational intentions are not trusted. The primary threat considered is the alteration of historical video records to evade liability, conceal events, or mislead forensic investigations.

To address this, PERQS maintains immutable blockchain-anchored hash commitments for recorded video segments. The threat model assumes that participants may attempt to modify previously stored footage after registration. PERQS does not prevent such modifications directly; instead, it ensures that any retroactive alteration of recorded data becomes detectable through hash verification against the immutable on-chain log.

3.4.1 Participant and Data-Level Threats

PERQS is designed to operate in an adversarial setting where correctness and integrity cannot be assumed.

- **Integrity of Recorded Data:** Recorded video data may be corrupted due to hardware failures, transmission errors, storage faults, or deliberate tampering. Such modifications can compromise the integrity and reliability of the footage, thereby weakening its evidentiary value in investigative or legal contexts.
- **Malicious Organizations:** A participant may intentionally provide falsified footage or manipulated analytical results to protect its interests, evade liability, or mislead other entities.
- **Byzantine Behavior:** Participants may behave arbitrarily or inconsistently, including selectively responding to queries, returning conflicting outputs, or colluding to bias the consensus process. The system assumes that a bounded subset of nodes may exhibit such behavior.

3.4.2 System Integrity Assumptions

While participants are not fully trusted, PERQS relies on certain foundational guarantees:

- The permissioned blockchain enforces authenticated membership and provides tamper-evident logging.
- The number of malicious participants does not exceed the tolerance threshold of the underlying consensus protocol.
- Standard cryptographic primitives for hashing and identity management remain secure.

Under these assumptions, PERQS aims to ensure integrity and fault tolerance without requiring trust in individual participants.

3.4.3 Privacy Considerations

Protecting video privacy is a central requirement in PERQS. Raw video feeds never leave the local domain of the camera owner. All analytics are performed locally, and only query outputs are shared for collaborative consensus. Although this architecture prevents exposure of full video streams, the outputs themselves—such as face detection results, license plate numbers, or vehicle identification metadata—may contain sensitive information. This introduces a limited and controlled form of privacy leakage.

However, the information disclosed is restricted to what is strictly necessary for consensus formation. The underlying raw footage remains private and under the control of the data owner. This design establishes a bounded and purpose-specific privacy trade-off that enables distributed, trust-aware video analytics while maintaining strong protection of original video content.

3.5 PERQS Design

PERQS connects public, private, and individual CCTV cameras into a contributory distributed network. Organizations that operate one or more cameras can participate and enable event querying across all participating feeds. This integrated approach provides broader coverage, particularly when incidents span multiple locations or require evidence from cameras owned by different entities.

A key design goal is seamless integration with existing CCTV infrastructure. PERQS preserves the current recording and storage architecture, allowing organizations to adopt the system without significant modifications. This minimizes deployment overhead and operational disruption.

PERQS follows a distributed architecture to preserve data ownership and privacy. Raw video feeds never leave the local domain of the camera owner. Each participant stores and analyzes its footage on local or privately managed servers. Only query results are shared for collaborative processing. This design eliminates centralization, distributes workload across participants, and naturally supports scalability as additional cameras and organizations join the network.

To handle unreliable or malicious participants, PERQS introduces *query consensus*. When multiple cameras cover the same location, their results are collectively evaluated to ensure reliability. This mechanism mitigates the impact of faulty devices or Byzantine behavior. For locations with limited or no camera coverage, PERQS employs *time-travel consensus*. In this

approach, nearby cameras are queried across relevant past and future time windows to identify causally related events. These auxiliary observations are aggregated to support consensus even in the absence of direct visual evidence.

The primary additional requirement for participants is the capability to perform video analytics. Each organization may deploy its own machine learning server locally or use remote infrastructure for processing. PERQS is compatible with diverse CCTV configurations, including local storage systems, cloud-based storage, and intelligent cameras with built-in analytics features.

To ensure integrity and fault tolerance, PERQS adopts a blockchain-based architecture. During recording, each video segment is hashed and committed to the blockchain, enabling later verification against tampering or corruption. Query execution and consensus decisions are recorded on the blockchain, providing transparency and immutability without exposing raw video data.

The system uses a permissioned blockchain model, where participants are authenticated organizations rather than anonymous entities. This structure supports role-based access control and aligns with the operational requirements of official authorities and institutional stakeholders. The distributed ledger also enhances resilience by tolerating individual node failures.

In summary, PERQS provides a scalable, privacy-preserving, and fault-tolerant framework for collaborative video querying across multiple organizations. The following sections describe the architecture and query execution mechanisms in detail.

3.5.1 PERQS Architecture

The design of PERQS uses a blockchain-based architecture composed of several vital components, as shown in Figure 3.3. At the system's core are the participating CCTV owner organizations, individuals, or groups that own one or more CCTV cameras. These organizations are connected to the PERQS network and deploy specific PERQS components alongside their CCTV video capture and storage service. The main components are:

- **PERQS Network:** A permissioned blockchain network connects all the participating CCTV owner organizations in PERQS. Access to the network is restricted to verified members, ensuring that only actual CCTV owner organizations can participate. Each organization has one or more blockchain peers, allowing for efficient and secure communication.

- **PERQS Client:** It is the blockchain peer that enables the participation of organizations in the PERQS network. It has multiple roles, such as committing the video hash to the blockchain, decoding and sending queries for execution, and reaching a consensus on query results. These functionalities are explained in more detail in the upcoming sections.

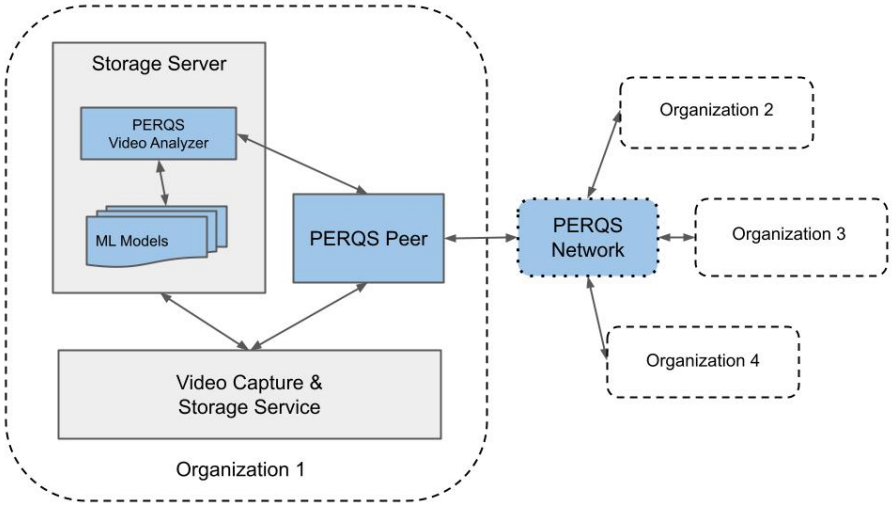

Figure 3.3: PERQS components in video capturing organizations. PERQS peer enables participation in the PERQS network and handle the hash commitment and query execution.

• **Video Capture & Storage Service:** In an organization, CCTV cameras are managed and stored using a central system called the video capture and storage service. Depending on the organization's preference, storage service can be on-premises or remotely. One of the critical features of PERQS is that it does not require any changes to the existing recording systems, but it must be integrated with the PERQS video analyzer.

• **PERQS Video Analyzer:** Responsible for performing analytical operations on recorded video streams. Queries are submitted to the analyzer, which decodes and executes them on individual video feeds. Each participating organization can deploy its video analysis servers either locally or remotely. The analyzer is designed to be modular and extensible, enabling easy integration of new video analytics models as they become available. This design ensures that the system remains adaptable to evolving technologies and continues to deliver accurate and reliable analysis results.

To summarize, PERQS is a decentralized system that enables organizations to collaborate in analyzing video footage without sharing their raw video feeds. Organizations must be able to conduct video analysis on their feeds using machine learning models. Participants are linked together using a blockchain network, and video committing ensures the integrity of the recordings, making them tamper-resistant.

3.5.2 Public Event Recording and Committing

PERQS uses blockchain-anchored cryptographic hashes to provide integrity verification for recorded video data. Once a video segment is registered, any later modification to that footage

can be detected through hash verification, making post-registration tampering computationally infeasible without detection.

However, the mechanism does not guarantee that the video was trustworthy at the time of capture. If a malicious participant alters footage before registration, the system cannot distinguish the modified content from a genuine recording because the blockchain only verifies consistency after commitment. Consequently, PERQS protects the integrity of stored evidence rather than the authenticity of the original capture process.

Overall, this approach strengthens accountability and traceability in collaborative surveillance systems by enabling independently verifiable evidence, even though source-level capture security remains outside the scope of the system.

3.5.3 Event Querying

PERQS allows only select entities, such as public authorities and participating organizations, to send queries and receive results. It utilizes a query language similar to SQL- PERQL(Public Event Recording and Querying Language), which allows users to specify a time and location of interest. This information is used to determine which cameras are likely to have captured the event in question. The query is then passed on to the camera owners, who perform video analytics on their saved video feeds and send the results back to the network. By combining the results from multiple organizations, PERQS forms a consensus result. This *query consensus* provides a comprehensive and accurate view of the event in the query. This consensus-based approach ensures that the system is reliable and fault-tolerant, even if one or more cameras are not working correctly or their video feeds are corrupted. In Section 3.6, we will provide a detailed explanation of the query execution engine in PERQS.

PERQS requires participants to register the cameras with their GPS location and field of view directions. This information narrows down the cameras of interest while performing a query execution. We use two oracles *1)* camera finder and *2)* timing oracle. An *oracle* is a trusted external entity that can provide off-chain data to the blockchain network[44]. Oracles can fetch data from external sources such as APIs or databases and make it available to the smart contracts running on the blockchain network. As the name suggests, a *camera finder oracle* is a tool that locates cameras in a specific geographic area based on GPS coordinates. For example, if a user inputs the GPS coordinates of a location, the camera finder oracle will return information about all the cameras that are close to that location. Oracle can use any technique to find a set of interesting cameras. In our case, it uses Euclidean distance and a more complex search circle expansion[84].

On the other hand, a timing oracle like [129] is used to synchronize the clocks of multiple

cameras. It is instrumental in a consensus setting, where multiple cameras may need to be time-synced to ensure accurate data collection and processing. When asked about two specific cameras, the timing oracle will return the time offset between the two cameras, which can then be used to adjust the clocks as needed. In our case, it is implemented using basic Network Time Protocol (NTP) [106] exchange between cameras.

3.6 Query Execution Engine

The query execution module handles user queries through three core components: the query language, decoding and execution, and video analyzers. Users interact with the system via the Public Event Recording and Querying Language (PERQL), which enables query formulation and submission to the PERQS client. The client, together with the video analyzer, decodes the query, identifies relevant video feeds, and performs the necessary analysis. Results from multiple video analyzers are then aggregated to generate the final output.

3.6.1 Query Language - PERQL

PERQL is a query language that provides flexible and customizable querying over distributed camera feeds. Unlike systems limited to a fixed set of predefined queries, PERQL enables users to express complex analytical queries using an SQL-like interface. PERQL extends FRAMEQL [77] with support for GPS locations, time intervals, and consensus-based operations across multiple camera feeds.

PERQL assumes that the output of video analytics pipelines is represented in a tabular format, enabling queries to be executed over structured metadata rather than raw video streams. In PERQL, a `MODEL` serves as a logical abstraction (or schema) representing the structured output of an underlying machine learning or computer vision analytic. While the underlying analytic performs object detection or classification, the PERQL `MODEL` keyword defines the corresponding table structure, including fields and data types, that makes the extracted information queryable. This abstraction enables a wide variety of queries to be executed efficiently over large-scale video datasets while decoupling query semantics from the implementation details of the underlying analytics engine.

Key operations supported in PERQL and their syntax are described below.

- **CREATE MODEL**

```
CREATE MODEL <model_name>
(
<data_field1> <data_type>,
<data_field2> <data_type>,
```

```
...)
```

A machine learning or vision model for video analysis is a trained algorithm capable of detecting and identifying objects, people, or activities within video streams. In PERQL, the `CREATE MODEL` command defines the output schema associated with a specific machine learning or vision analytic. While the underlying model performs the actual detection or classification, the PERQL command establishes the relational schema—i.e., the output data fields and their types—that allows the analytic results to be queried as a table.

When a new analytic model is introduced into PERQS, it can be distributed across participating organizations through the query infrastructure, enabling servers to support updated video analytics capabilities. The *data_field*s correspond to the structured outputs generated by the underlying analytic model.

- **ALTER MODEL**

```
ALTER MODEL <model_name>
ADD <data_filed> <data_type>
```

When there is a change to an existing model definition, the corresponding query propagates the update across participating organizations. Modifications to the underlying machine learning or vision algorithm may necessitate updates to its PERQL schema. For example, a *vehicleDetector* model may initially expose fields such as *vehicleType* and *vehicleModel*. If the underlying analytic is later upgraded to support color detection, an additional *vehicleColor* field must be added to the schema. In such cases, the `ALTER MODEL` command is used to update and distribute the revised schema across all organizations.

- **SELECT**

```
SELECT <fields>
FROM <model_name>
WITH <feature_parameters> -- optional
AT <gps_loc>
WHERE <conditions> -- optional
TIME <start_time> TO <end_time>
```

A SELECT query is used to extract information from the collective video database. The video feeds are narrowed down using the GPS location and time data. On the video feeds, the owner organization's video analyzer server applies an ML/vision model to perform analysis and create a table with the results. The query conditions are then used to filter out unwanted results.

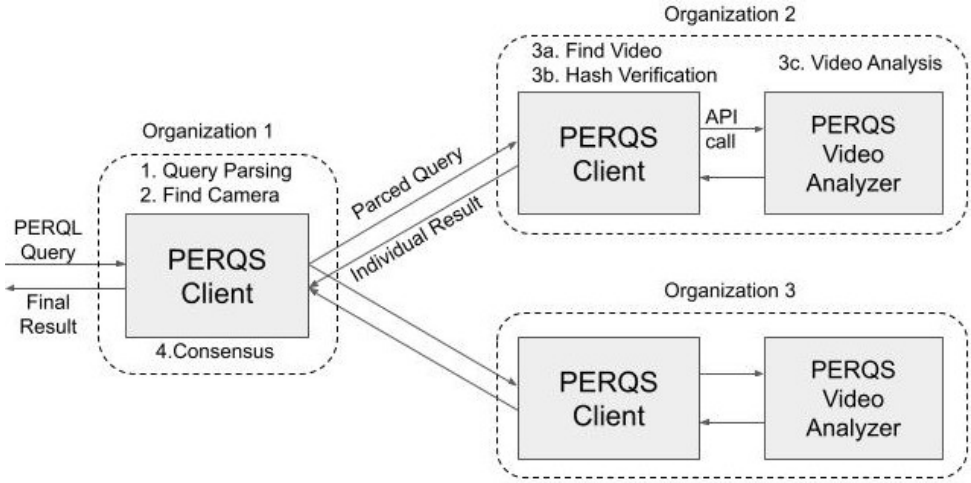

Figure 3.4: PERQL query execution stages. Organization 1 initiates a query execution. Sends the parsed query to two owner organizations, 2 and 3.

Using the WITH keyword, we pass optional parameters to any models if required. Finally, the results from multiple cameras are combined to form the final output of the SELECT query.

Using a query language in PERQS makes the system highly expandable, providing numerous possibilities for extracting information from the video database. The model plug-and-play design ensures that new updates and techniques in the field of computer vision can be quickly integrated into the system. The language used in the system, similar to SQL, makes it easy for developers to design queries for natural language requests. PERQL provides a more intuitive, user-friendly user experience and powerful querying capabilities. Make the system more accessible and efficient in providing accurate information from the video footage.

3.6.2 Query Decoding and Execution

Interpreting and executing queries is at the heart of the PERQS system. The entire execution engine is spread across clients, peers, and video analytic servers. When a query is initiated, it reaches the client and is forwarded to the organizations for video analysis. Finally, the results are sent back, and a consensus step is implemented to ensure that false results caused by malicious actors are eliminated. A single PERQS query execution consists of four stages, as shown in Figure 3.4.

① **Query parsing:** Decode the keywords and parameters from the query string. If the GPS location or time details are not in the query, we abort the execution.

② **Find the camera:** Identify the camera sets of interest based on the GPS location details. A *'camera-finder'* oracle returns the camera ID given GPS location and range. Each organization

	Output Type	Consensus Rules	Error handling/policies
1	String	Equals	Majority, at least n
2	Set of strings	Set Intersection / Set Union	Strings present with majority or n of the results
3	Range (time, integer, float)	Overlapping range	Floor or ceil of overlapping
4	Number(float/int)	Equals	$\pm 5\%$ is considered as equal (x%)
5	Set of numbers(float/int)	Set Intersection	majority, $\pm 5\%$ is considered as equal
6	Color (hex string)	Equals	$\pm 5\%$ in each RGB is considered as equal
7	Boolean(true/false, yes/no)	Equals	Majority, at least n

Table 3.1: Example Fuzzy Consensus Rules for Different Data Types.

registers the camera location details to the *camera-finder* oracle at the time of joining or when a new camera is added. Once we identify the cameras, send the decoded query to each owner.

③ **Owner query execution:** Participating organizations find the corresponding video using the time parameter passed with the query. Then, the client verifies the hash of the video with the committed hash in the blockchain. If the hashes match, an API call is made to the video analytics server with the video, *video_id*, and decoded query parameters. It analyses the video and returns the result as a table per the query options. This result is sent back.

④ **Consensus:** Combine all the tables received using the consensus rules and policies. Example consensus rules for commonly used data types are shown in Table 3.1. PERQS supports fuzzy consensus evaluation to reconcile non-exact matches across heterogeneous camera feeds and analytics models. For example, numerical values are matched using a $\pm 5\%$ tolerance, color values are compared using bounded RGB variance, temporal and numerical ranges are reconciled through overlap analysis, and set-valued outputs are merged using intersection/union operations with majority thresholds. New data types and corresponding consensus policies can be incorporated as additional analyzers or ML models are integrated into the system. Policies include majority, at least n cameras, and at least n organizations. Default policies configured by the system can also be overridden through query parameters when required. Instead of relying on the analysis of a single camera, the consensus algorithm requires a minimum number of cameras, t, to participate in reaching a collectively verified decision.

3.6.3 Fuzzy Evaluation Procedure

PERQS does not perform strict bitwise equality checks during consensus evaluation. Instead, the system uses a configurable fuzzy matching procedure to account for natural variations across distributed camera feeds and independently generated analytics results. This is necessary because cameras in a contributory surveillance network often differ in resolution, viewing angle,

exposure settings, lighting conditions, and calibration. As a result, the same real-world object or event may be represented differently across sensors even when all cameras are functioning correctly.

For color attributes represented as hexadecimal RGB values, detections are evaluated using configurable channel-wise variance thresholds rather than exact matches. This allows the system to reconcile visually similar detections captured under different illumination conditions. For example, the same blue vehicle may appear as slightly different hex values across cameras while still representing the same object. The percentage thresholds shown in Table 3.1 are illustrative examples; acceptable fuzzing limits can be configured during PERQS deployment based on the characteristics of the camera network and application requirements.

Similarly, numerical attributes such as integer or floating-point values are evaluated using tolerance-based comparisons instead of strict equality. The acceptable deviation range is configurable and can be tuned depending on the expected variability of analytics outputs such as object dimensions, speed estimates, or confidence scores.

For temporal or numerical ranges, PERQS applies overlap-based reconciliation rather than exact boundary matching. Consensus is established by computing the overlapping interval across observations and taking the floor/ceiling bounds of the shared region. This enables robust agreement despite timestamp drift or range estimation differences between cameras.

For set-valued outputs, such as lists of detected objects or attributes, PERQS uses set intersection or union operations combined with majority thresholds to determine consensus. This allows the system to reconcile partially overlapping detections from multiple cameras while filtering out inconsistent or spurious observations.

By utilizing this fuzzy logic, PERQS overcomes the limitations of traditional relational databases (like PostgreSQL), which would fail to return results due to strict hex-string mismatches. This approach significantly improves Recall and ensures that valid evidence is not discarded due to minor sensor variations.

3.6.4 Video Analyzer

In PERQS, a video analyzer refers to a set of vision-based tools and modules to analyze video feeds and detect events, objects, or features. We consider the video analyzer module as a function that takes a video feed and specific parameters as input and returns a table as output. The columns of this table represent the output parameters of the model used, and each row represents an event or object that the vision model detected. This approach allows for easy and efficient processing of large amounts of video data, making it possible to extract valuable information from the feeds quickly. An example is an object Detector ML model, which will

detect objects in each video frame.

3.6.5 Time-Travel Consensus

PERQL query execution typically reaches consensus when multiple cameras monitor the same location and a majority behave honestly. However, consensus may not be achievable when (i) no cameras are present at the specified GPS location, or (ii) the number of available cameras is insufficient to satisfy the configured consensus policy. To address these limitations, we introduce *time-travel consensus*.

Time-travel consensus extends the scope of analysis beyond a single location and timestamp. Instead of relying solely on direct visual evidence at the queried location, the system performs a **spatiotemporal search** across nearby cameras and adjacent time intervals. The central observation is that real-world events are rarely isolated; they are often causally linked to activities that occurred earlier or later at neighboring locations. Thus, even when direct footage is unavailable, supporting evidence may exist elsewhere in the network.

3.6.5.1 Functional Query Adaptation

Within the PERQS framework, time-travel consensus operates as a functional query adaptation mechanism. Consider a user submitting a query regarding an accident at a specific location. If the accident occurred in a camera blind spot, the system cannot rely on a standard accident-detection model because the collision itself was never captured. In such cases, a direct accident query would fail to produce results.

Rather than terminating unsuccessfully, PERQS reformulates the query to search for *corroborative evidence*. The system shifts from detecting the collision itself to identifying causally linked activities, such as:

- Vehicle speed prior to the estimated time of collision,
- Vehicle direction and trajectory,
- Vehicle presence at adjacent junctions.

This reformulation is facilitated by the use of *plug-and-play* machine learning models, which are used to extract granular spatiotemporal data from auxiliary video feeds. Specifically, these models are utilized for speed estimation, vehicle trajectory analysis, and cross-camera vehicle re-identification (Re-ID) to infer the likelihood of an event occurring in a blind spot.

For example, if only an accident detection model is available, consensus cannot be achieved directly in a blind spot where no camera has visibility. However, by executing speed and direction estimation models on footage from nearby cameras, the system can infer whether a collision likely occurred. In this way, time-travel consensus reformulates a failed direct query

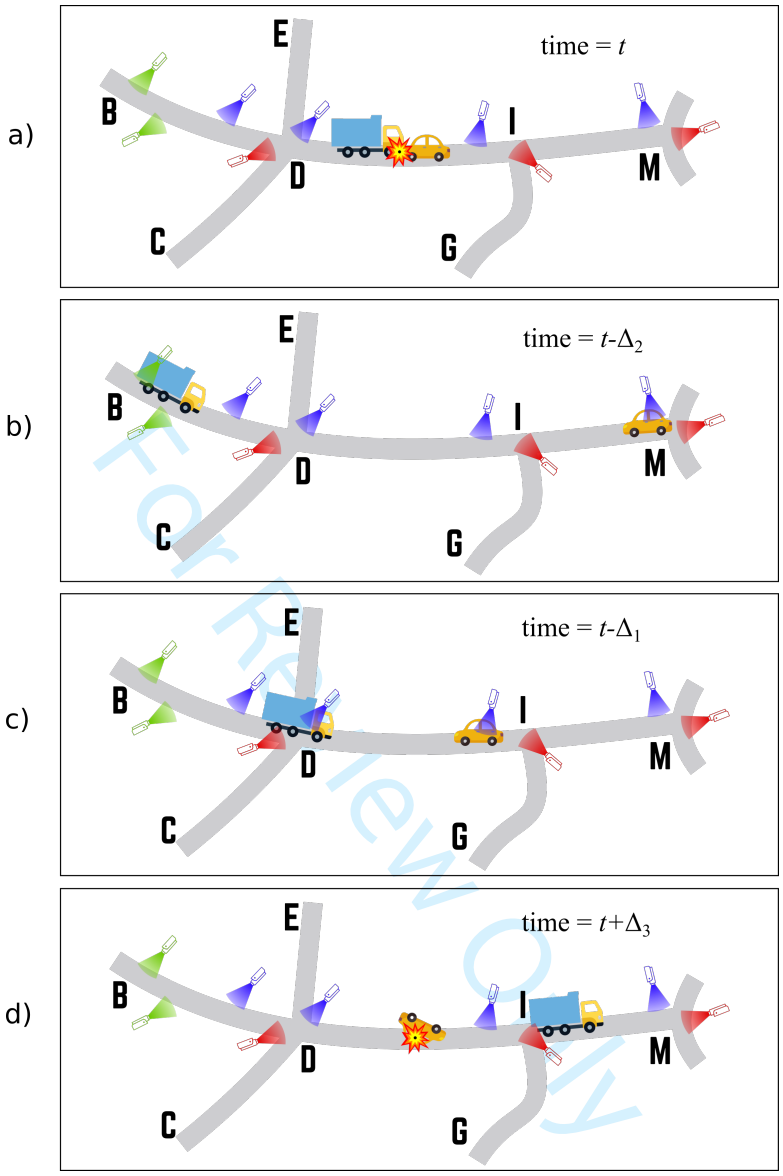

Figure 3.5: Example 2 Section 3.3 illustration for time travel consensus. a) An accident occurred. b) Positions of the vehicles at $time = t - \Delta_2$. c) Positions at $time = t - \Delta_1$. d) Hit and run. Position at $time = t + \Delta_3$. Only the cameras which might capture the vehicles are shown.

into an indirect inference problem.

3.6.5.2 Illustrative Example

Consider the scenario shown in Figure 3.5. A truck from junction B and a car from junction M collide in a blind region between junctions D and I. Although no camera directly records the impact, both vehicles are captured shortly before and after the accident at junctions B, D, M,

and I.

At times $t - \Delta_2$ and $t - \Delta_1$, the vehicles are visible at upstream junctions. At time $t + \Delta_3$, one vehicle may be observed leaving the scene, which is particularly relevant in hit-and-run scenarios. Instead of searching for the collision itself, PERQS executes speed estimation and trajectory analysis models at neighboring junctions within carefully selected time windows:

$$(t - \Delta) \pm \theta \quad \text{and} \quad (t + \Delta) \pm \theta,$$

where θ defines the temporal search interval.

By correlating these observations across multiple cameras, the system can estimate vehicle speeds, detect overspeeding, and infer causal responsibility. If multiple cameras independently confirm abnormal behavior, consensus can be reached even without direct footage of the accident.

3.6.5.3 Core Principles

The time-travel consensus mechanism is guided by the following principles:

• **Spatiotemporal Context:** Events are interconnected across time and geography. A collision at one location is often preceded or followed by observable behaviors elsewhere.

• **Indirect Evidence Aggregation:** Consensus is achieved by aggregating supporting evidence from nearby cameras rather than requiring direct visual confirmation.

• **Plug-and-Play Model Integration:** Machine learning and vision models are treated as modular components that provide the evidentiary inputs required for consensus. These models extract granular spatiotemporal features—such as predicted vehicle positions, speeds, trajectories, and cross-camera identities—which are then aggregated by the time-travel consensus mechanism to perform indirect inference. As improved analytics models become available (e.g., enhanced tracking or trajectory prediction), they can be integrated without modifying the underlying consensus framework.

3.6.5.4 Operational Procedure

Time-travel consensus is triggered when the camera-finder oracle returns fewer cameras than required by the configured consensus policy.

① The system expands the spatial search radius R to identify additional nearby cameras that may have captured causally related events.

② A corresponding temporal offset Δ is computed, often as a function of R, based on the expected vehicle travel time within that radius.

③ The **original query is reformulated** if direct evidence is unavailable. For example, an *accident detection query* may be transformed into one or more auxiliary queries such as:

- Speed estimation queries at upstream junctions,
- Vehicle trajectory or direction estimation queries,
- Vehicle presence verification queries within the estimated time window.

The reformulated queries target causally linked activities rather than the primary event itself.

④ These modified queries are executed on the newly identified camera feeds within the estimated time interval $(t - \Delta) \pm \theta$ and $(t + \Delta) \pm \theta$.

⑤ The results from all participating cameras are aggregated and correlated. The system searches for consistent patterns across cameras, such as overspeeding vehicles, matching trajectories, or temporal alignment of vehicle appearances.

⑥ If a sufficient number of independent cameras provide mutually consistent corroborative evidence, the consensus policy is satisfied and a consensus is established regarding the event-of-interest.

The parameters R and Δ are determined during system deployment through empirical testing. In dense urban areas, a small R may suffice due to high camera density. In sparse regions, a larger radius is required. If sufficient cameras are not identified, R may be expanded iteratively.

It is important to note that consensus is not achieved by detecting the original event directly, but by validating a set of causally consistent auxiliary observations derived from the modified queries. In this manner, the system converts a failed primary query into a structured multi-query inference process that enables consensus even in the absence of direct visual evidence.

3.6.5.5 Limitations

Time-travel consensus is not universally applicable. First, some events may not produce observable causal traces at nearby cameras, making indirect reconstruction impossible. Second, cameras within the expanded radius R may be irrelevant to the event-of-interest and provide no supporting evidence. Third, the mechanism depends on the availability of appropriate machine learning models. If only an accident detection model is available and no camera captured the collision, consensus cannot be reached. Models capable of speed estimation, direction inference, number plate detector, vehicle tracking, color detector etc. are important for indirect reasoning.

The precision of the time-travel inference leverages the performance of specific vision models. While the consensus logic itself is a structural estimation, models for speed estimation, direction inference, and tracking are essential for providing the high-quality data necessary for

Figure 3.6: CityFlowV2 [49]: Case study 1: time sync screenshots of 5 cameras when a blue SUV truck crossing the junction at 11:30:05.50.

accurate indirect reasoning. Although these analytics are integrated in a plug-and-play manner, improvements in computer vision and tracking technologies will directly enhance the robustness and accuracy of the consensus mechanism.

3.7 Experimental Setup

Our PERQS prototype was built on Hyperledger Fabric v2.2 [66] and deployed on a Kubernetes cluster [89]. The video analytics servers were developed using the Python Django framework, utilizing Django v4.1.4, Python v3.10.6, and TensorFlow v2.8.0. In our implementation, the PERQS network is a Hyperledger Fabric network, with the Fabric client and Fabric peer performing the role of the PERQS client. Smart contracts were used to implement hash committing and query executions.

3.7.1 PERQS Experiments

We set up a Fabric-based testbed for our experiments, consisting of three organizations. There is one orderer node, and each organization has a single peer node. Our Kubernetes cluster setup contains six nodes. All nodes run as a virtual machine and are allocated vCPUs of Intel(R) Core(TM) i9-7920X CPU, running at 2.90GHz. Fabric components are deployed as Kubernetes pods. We have set up our video analyzer as a python-django server, which utilizes GeForce GTX 1080 Ti GPU available in the server.

We have used the AICITY21 benchmark (CityFlowV2) [49], [35], captured by 46 cameras in a real-world traffic surveillance environment. A total of 880 vehicles are annotated in 6 different scenarios. There are 215.03 minutes of videos in total. We have identified eight junctions where we have multiple camera recordings.

3.8 Evaluation

The primary objective of the evaluation is to validate the effectiveness, and robustness of the PERQS consensus mechanisms under realistic traffic surveillance conditions. The evaluation focuses on three key aspects: (i) correctness of collaborative query execution, (ii) robustness against faulty or malicious participants, and (iii) the performance overhead introduced by distributed consensus and verification.

Although PERQS ensures post-registration integrity through blockchain-backed hash commitments, a formal cryptographic evaluation of the underlying blockchain platform or the machine learning models is outside the scope of this chapter. Instead, the evaluation concentrates on the behavior of the distributed query execution engine and the reliability of the consensus mechanisms under adversarial and incomplete observation conditions.

3.8.1 Benchmarking Framework

To systematically evaluate PERQS, we constructed a benchmark framework using the CityFlowV2 surveillance dataset [49]. From the dataset, we identified eight traffic locations containing multiple cameras with either overlapping or partially connected fields of view. These junctions provide a realistic environment for evaluating collaborative video analytics under heterogeneous observation conditions.

Rather than relying solely on manually selected examples, we generated 40 randomized PERQL queries by varying vehicle attributes such as color and type . Example queries include searches such as *Find a white car*, *Count cycle*, *Find a silver truck*, or *Count blue SUV*. This benchmark enables repeatable and large-scale evaluation of both correctness and performance across diverse traffic scenarios. Table 3.2 listed the 40 randomly generated queries. Ground Truth denotes the manually verified vehicle count over a two-minute interval. Accuracy is measured on a per-query basis: a query is considered correct only if the system correctly detects and classifies all vehicles present in the ground truth. Any missed or misclassified vehicle results in the query being counted as incorrect. The same evaluation criterion is applied to both individual cameras and PERQS Table 3.3 listed the available camera IDs according to CityFlowV2 dataset and query result summary per location.

The benchmark evaluates PERQS using plug-and-play analytics models integrated into the PERQ video analyzer framework. We used YOLOv3 [153] as the primary object detector to identify vehicles frame by frame. The detector outputs structured records containing attributes such as *vehicleID*, *timeAppeared*, *vehicleType*, and *vehicleColor*. These independently generated outputs are subsequently reconciled using the PERQS consensus mechanisms.

QID	Query	J	Ground Truth	C1	C2	C3	PERQS 3_Cams
1	White Car	1	3	2	3	3	3
2	Blue Truck	1	2	2	2	2	2
3	Red Car	1	4	2	3	4	3
4	Cycle	1	2	2	3	2	2
5	Blue Car	1	3	3	3	4	3
6	Black Car	1	2	1	2	2	2
7	Blue Truck	1	1	1	1	2	1
8	Red Truck	1	4	4	4	2	4
9	Black Car	1	3	2	3	3	3
10	Blue Truck	1	2	2	2	2	2
11	Cycle	2	2	1	1	2	1
12	Red Car	2	3	3	3	3	3
13	Black Car	2	1	2	1	1	1
14	Blue Truck	2	4	4	3	4	4
15	White Truck	2	2	2	2	3	2
16	Blue Truck	2	3	3	3	2	3
17	Black Car	2	4	4	3	4	4
18	Red Car	2	5	5	5	5	5
19	Black Car	3	3	2	2	3	2
20	White Car	3	2	2	2	3	2
21	Red Car	3	1	1	1	2	1
22	Cycle	3	2	2	2	1	2
23	Red Car	3	3	3	3	3	3
24	Green Car	3	2	1	2	1	1
25	Blue Car	3	3	2	3	3	3
26	Green Car	3	4	4	3	4	4
27	White Car	3	2	2	2	2	2
28	Blue Truck	4	3	3	3	2	3
29	White car	4	3	3	4	3	3
30	Blue Car	4	2	1	2	2	2
31	White Car	5	4	2	4	3	3
32	White Truck	5	3	3	3	3	3
33	Green Truck	5	2	2	2	2	2
34	White car	6	4	4	3	3	3
35	Blue Car	6	3	3	3	3	3
36	Black Truck	7	2	2	2	2	2
37	Cycle	7	4	3	4	4	4
38	White Car	7	3	3	2	3	3
39	Gray Car	8	3	3	3	2	3
40	Blue Truck	8	1	1	2	2	2
Accuracy(%)				**70**	**72.5**	**65**	**82.5**

Table 3.2: Results of 40 randomly generated benchmark queries executed using three participating cameras.

3.8.2 Benchmark Evaluation

We evaluated all 40 benchmark queries under two different deployment configurations. In the first configuration, consensus was performed using only three cameras per junction. In the second configuration, all available cameras covering the junction and nearby roads participated in the consensus process. This setup allowed us to evaluate how the number of participating cameras affects query reliability, fault tolerance, and overall detection accuracy.

For the first three junctions, the available camera feeds provided overlapping views of the same intersection. Therefore, we applied a simple majority consensus policy. The consensus rule required at least 50% of the participating cameras to report a matching detection before it was included in the final consensus result. For example, if four cameras participated in the query execution, the vehicle was accepted into the final result when at least two cameras independently detected it. The threshold can be adjusted depending on deployment requirements, camera density, and the desired Byzantine fault tolerance guarantees.

The remaining five junctions represented route segments with partial or disconnected coverage rather than fully overlapping intersections. In these locations, cameras monitored different portions of the road network, making direct majority comparison insufficient. PERQS therefore utilized time-travel consensus to reconstruct detections using neighboring cameras and temporal offsets. Although a vehicle might not appear directly in the queried location, it would often be observed by nearby cameras while traversing the route. By correlating vehicle type, color, and appearance time across these cameras, PERQS successfully reconstructed missing observations and inferred vehicle presence even when direct visual confirmation from the queried location was unavailable.

In Table 3.3, *PERQS 3_Cams* represents the results obtained using only three participating cameras, while *PERQS All_Cams* represents the results obtained using all available cameras. The comparison demonstrates that increasing the number of participating cameras improves both reliability and resilience to noisy detections. Additional camera views provide stronger redundancy, reduce the influence of faulty or malicious nodes, and improve the robustness of the final consensus result. The results show that collaborative consensus substantially improves detection accuracy compared to relying on individual camera feeds. Using only three cameras, PERQS achieved an average accuracy of 82.5%. When all available cameras participated in the consensus process, the accuracy increased to 92.5%. These results demonstrate the effectiveness of distributed consensus in improving query reliability across heterogeneous and partially overlapping surveillance deployments.

Later in this section, we present three detailed case studies illustrating the complete end-

J	Camera Ids	# of Queries	C1	C2	C3	PERQS 3_Cams	PERQS All_Cams	Consensus Type
1	1,2,3,4,5	10	6	8	6	8	9	Simple Majority
2	6,7,8,9	8	8	7	7	8	8	Simple Majority
3	36,38,39,40	9	7	7	7	8	8	Simple Majority
4	33,34,35,36,37	3	1	1	1	2	3	Time-Traval
5	29,30,31,32	3	2	2	2	2	2	Time-Traval
6	22,23,24,25,26	2	1	1	1	1	2	Time-Traval
7	18,19,20,21	3	2	1	1	2	3	Time-Traval
8	10,11,12,13,14	2	1	2	1	2	2	Time-Traval
Accuracy			**70%**	**72.5%**	**65%**	**82.5%**	**92.5%**	
Precision			**79.4%**	**77.4%**	**73.1%**	**84.6%**	**90.6%**	
Recall			**70.6%**	**75.2%**	**69.7%**	**80.7%**	**88.1%**	
F1-Score			**0.748**	**0.763**	**0.714**	**0.826**	**0.893**	

Table 3.3: Query execution performance across eight traffic junctions. Values denote the number of correct query responses generated by individual camera configurations (C1, C2, C3) and 3-camera (PERQS 3_Cams) and all-camera (PERQS ALL_Cams) consensus results.

Figure 3.7: Altered Camera 1 image, brightness reduced by 20% and contrast increased by 20%.

to-end query execution workflow and demonstrating how PERQS performs majority consensus, time-travel consensus, and Byzantine fault tolerance in practice.

3.8.3 Adversarial Experiments

To evaluate the robustness of PERQS under realistic deployment conditions, we designed a series of adversarial experiments simulating faulty sensors, unreliable analytics models, and Byzantine participants.

Data Faults: We first simulated degraded camera operating conditions using multiple fault injection strategies. Specifically, we randomly dropped 20% of the frames from the video stream and altered the visual characteristics of the feed by reducing brightness and increasing contrast, as illustrated in Figure 3.7. These modifications emulate real-world deployment issues such as network instability, motion blur, low-light environments, sensor degradation, or poor camera calibration.

Experiment by introducing faults in camera 1 (C1)	Camera 1				PERQS 3_Cams				PERQS All_Cams			
	A	**P**	**R**	**F1**	**A**	**P**	**R**	**F1**	**A**	**P**	**R**	**F1**
Without any fault simulation	70.0%	79.4%	70.6%	0.748	82.5%	84.6%	80.7%	0.826	92.5%	90.6%	88.1%	0.893
Camera Fault (C1)	55.0%	64.1%	52.3%	0.576	77.5%	78.4%	76.9%	0.777	90.0%	88.7%	86.2%	0.874
YOLOv3 (Decrease Confidence C1)	42.5%	68.7%	82.6%	0.750	85.0%	82.8%	88.1%	0.853	92.5%	88.4%	90.8%	0.896
YOLOv3 (Increase Confidence C1)	45.0%	82.4%	51.9%	0.636	75.0%	89.4%	77.1%	0.828	87.5%	91.1%	85.2%	0.880
Reporting 50% incorrect Values	32.5%	39.6%	35.2%	0.373	70.0%	81.3%	71.6%	0.761	85.0%	85.6%	81.7%	0.836
Reporting 100% incorrect Values	00.0%	5.4%	4.6%	0.049	57.5%	75.5%	57.1%	0.651	77.5%	83.3%	77.9%	0.806

Table 3.4: Evaluation benchmark by varying Camera 1 to simulate different fault and adversarial conditions. The columns corresponding to Accuracy(A), Precision(P), Recall(R) and F1-Score(F1).

The results in Table 3.4 show that faults affecting a single camera significantly reduce the accuracy of that participant. However, the impact on the final PERQS consensus remains comparatively limited because the remaining cameras continue to contribute correct observations during consensus formation.

Poor Classifiers: To simulate unreliable analytics models, we intentionally modified the confidence thresholds of the YOLOv3 detector. Lower confidence thresholds caused the detector to produce a larger number of noisy detections and false positives, while higher thresholds increased the probability of missing valid detections. These experiments evaluate whether the PERQS consensus mechanisms can compensate for low-quality machine learning outputs generated by individual participants.

Interestingly, decreasing the YOLOv3 confidence threshold increased the accuracy of *PERQS 3_Cams*. Although the detector generated more false positives, it also detected additional valid vehicles that would otherwise have been missed. Since consensus is reached collectively across cameras, many incorrect detections were filtered out during aggregation, while valid detections reinforced each other across participants.

In contrast, increasing the confidence threshold reduced recall because valid vehicle detections were discarded more aggressively. This reduced the probability of achieving consensus among the participating cameras and consequently lowered the overall PERQS accuracy.

Byzantine Actors: We additionally designated a subset of cameras as malicious participants. These nodes intentionally returned incorrect results, including:

- Reporting a *car* as a *truck*
- Returning incorrect color codes
- Omitting valid detections

We evaluated two Byzantine scenarios: (i) cameras reporting incorrect values 50% of the time, and (ii) cameras reporting incorrect values 100% of the time. Table 3.4 summarizes the results of these experiments.

When Camera 1 intermittently reported incorrect values, the accuracy of *PERQS 3_Cams* dropped because consensus could only be formed reliably when Cameras 2 and 3 correctly identified the target vehicle. When Camera 1 reported incorrect values continuously, the degradation became more severe, as one-third of the consensus participants consistently behaved maliciously.

However, the *PERQS All_Cams* configuration maintained comparatively high accuracy even under fully malicious behavior. The larger number of participating cameras provided stronger redundancy and enabled the system to suppress faulty observations through majority agreement. This demonstrates that increasing the number of independent participants significantly improves Byzantine resilience.

Overall, these experiments demonstrate that PERQS remains robust even when a subset of participants behaves maliciously or unreliably. As long as an honest majority exists, the consensus mechanism continues to produce correct and collectively verified query results despite noisy detections, degraded video quality, or adversarial participants.

3.8.4 Accuracy and Performance

We evaluated PERQS using both accuracy and performance metrics to quantify the effectiveness of the consensus mechanisms and the overhead introduced by distributed query execution. To assess the reliability and completeness of the query results, we use four standard evaluation metrics:

- Accuracy (A): The ratio of correctly answered queries to the total number of benchmark queries executed.
- Precision (P): The ratio of correctly identified events (True Positives) to the total number of returned results (True Positives + False Positives). In PERQS, Precision reflects the ability of the consensus mechanism to suppress noisy or malicious detections.
- Recall (R): The ratio of correctly identified events (True Positives) to the total number of actual events present in the ground truth video (True Positives + False Negatives). Recall measures the ability of PERQS to recover events missed by individual cameras due to blind spots, failures, or incomplete coverage.

- F1-Score (F1): The harmonic mean of Precision and Recall, providing a balanced measure of overall query reliability.

Accuracy Metrics: During the benchmark evaluation, we measured the Accuracy (A), Precision (P), Recall (R), and F1-score (F1) of individual camera outputs and compared them against the final consensus results generated by PERQS, as summarized in Table 3.3 and Table 3.4. Among the 40 benchmark queries, 28 queries used majority consensus across overlapping camera feeds, while the remaining 12 queries required time-travel consensus using spatiotemporal reconstruction across neighboring cameras.

The results show that individual cameras are highly sensitive to degraded video quality, noisy detections, and Byzantine behavior. For example, when a camera continuously reported incorrect values, its standalone accuracy dropped to 0%. In contrast, PERQS maintained significantly higher reliability through collaborative reconciliation. Using only three cameras, PERQS achieved up to 82.5% accuracy and an F1-score of 0.826 under normal conditions, while using all available cameras further improved the results to 92.5% accuracy and an F1-score of 0.893.

Majority consensus improved robustness by filtering inconsistent detections through collective agreement across multiple cameras. Even under adversarial conditions where one camera intentionally produced incorrect outputs, PERQS All_Cams maintained 77.5% accuracy and an F1-score of 0.806, demonstrating resilience against Byzantine participants. Similarly, under degraded camera conditions and noisy YOLOv3 detections, the consensus mechanism continued to preserve high Precision and Recall by suppressing isolated false positives.

Performance Metrics: Query execution in PERQS can be divided into four major stages: query planning, query parsing, video analytics execution, majority or time-travel consensus. To measure query planning, parsing, and consensus overheads, we simulated the process 1000 times and computed the average. Video analytics performance was measured using the benchmark framework consisting of 40 camera recorded videos, each averaging three minutes at 10 fps.

Query Planning	Query Parsing	Video Analytics	Majority Consensus	Time-travel Consensus
31.5ms	230.5ms	1.72s	47.15ms	89.44ms

The evaluation results show that video analytics dominates the overall query execution cost, while consensus formation and query coordination introduce comparatively small overheads. Processing a three-minute video segment using YOLOv3 requires approximately four minutes

for object detection and an additional minute for attribute extraction and color analysis, resulting in an average processing cost of 1.72 seconds per second of video. In comparison, majority consensus requires only 47.15ms on average, while time-travel consensus requires 89.44ms due to additional trajectory reconstruction and temporal correlation operations. These results demonstrate that the primary bottleneck in PERQS is machine learning inference rather than distributed consensus. Unlike traditional databases such as PostgreSQL or DuckDB, which optimize query latency over centrally stored and trusted data, PERQS focuses on producing collectively verified results across multiple independent organizations where raw video feeds cannot be centrally aggregated due to privacy and trust constraints. Consequently, the dominant overhead arises from distributed video verification and collaborative analytics rather than query parsing or table lookup operations.

3.8.5 Case Studies

To demonstrate detailed query execution behavior, we present three representative case studies illustrating majority consensus, time-travel consensus, and Byzantine robustness. We define a model named *vehicleDetector*, which uses YOLOv3 [153] to detect vehicles such as cars, trucks, bikes, and SUVs on a frame-by-frame basis. Overlapping detections are merged across frames to produce stable vehicle observations, while cropped vehicle images are further analyzed to determine vehicle color and model information. The resulting analytics pipeline generates structured outputs containing attributes such as *vehicleID*, *timeAppeared*, *vehicleType*, *vehicleColor*, *vehicleModel*, and *vehicleModel*, which are subsequently used for collaborative consensus formation across distributed cameras.

Case Study 1 - Majority consensus: Find a blue truck passed through junction one at around 11:30 AM. This case study is a simplified version of Example 1 described in Section 3.3 to demonstrate the majority consensus. Consider Figure 3.6; we have five views of the junction from five different cameras. Representing the query in our PERQL as follows

```
SELECT timeAppeared, vehicleType, vehicleColor
FROM vehicleDetector
AT 42.525678, -90.723601
WHERE vehicleType="truck", vehicleColor="#0000FF"
TIME 2023-01-25 11:30:00.005 TO 2023-01-25 11:35:00.005
```

① The query planning stage. Find the cameras available at junction one(42.525678, -90.723601) using the *camera-finder* oracle.

② Forward the query to each of the camera owners.

timeAppeared	vehicleType	vehicleColor
Camera 1 result table		
11:30:04.90 11:30:07.10	truck	#012AB4
11:30:36.30 11:30:38.70	truck	#1336B8
11:32:08.40 11:32:09.90	truck	#15247F
11:33:12.60 11:33:14.50	truck	#3655AA
Camera 2 result table		
11:30:04.30 11:30:06.70	truck	#103BB8
11:30:35.90 11:30:37.70	truck	#1538B9
11:32:07.90 11:32:08.60	truck	#20247C
11:33:12.20 11:33:14.30	truck	#26456A
Camera 3 result table		
11:30:05.10 11:30:17.30	truck	#002CAD
11:30:36.50 11:30:48.70	truck	#1A35B4
11:32:08.20 11:32:19.30	truck	#25249B
11:33:12.90 11:33:23.20	truck	#36559A
Camera 4 result table		
11:30:01.80 11:30:06.80	truck	#003AB3
11:30:31.40 11:30:38.40	truck	#1254A4
11:33:09.20 11:33:14.10	truck	#36557A
Camera 5 result table		
11:30:04.80 11:30:09.20	truck	#001261
11:33:12.30 11:33:16.60	truck	#26356A
Final result table after consensus		
11:30:05.10 11:30:06.70	truck	#032CA2
11:30:36.50 11:30:37.70	truck	#1234B4
11:32:07.90 11:32:08.60	truck	#15245F
11:33:12.90 11:33:14.10	truck	#36558A

Table 3.5: Case study 1: reply from five cameras available at junction one and the final result after majority consensus.

③ Each camera owner executes the query.

- From the time information, identify which stored video feed got the visuals
- Verify the hash commitment to check whether the video has been tampered with. In case of hash mismatch, abort.
- Call the *vehicleDetector* video analyzer with the video feed and fields required. Object detector returns a table with *vehicleID*, *timeAppeared*, *vehicleType*, *vehicleColor*, *vehicleModel*, and *featureVector*. Execute the condition to filter out rows not required. *vehicleType* and color similarity in this case. Filtered tables for these five cameras is shown in

Figure 3.8: Case study 2: cycle captured by camera 10, 11, 12, 13, 15 for the find cycle query at different times.

Table 3.5. Send the table back to the querier.

④ Collect the results from all camera owners. Use the time sync oracle to synchronize the tables. Then, compare the tables to construct the final table. If the majority of the cameras spotted a car, it would be there in the final list. Table 3.5 shows the query execution results.

Fuzzy Color Matching Logic: In real-world traffic surveillance, a single "blue" vehicle will rarely produce an identical hex-code signature across different cameras. As observed in Table 3.5, individual camera outputs for the same truck varied across a spectrum of blue shades (e.g., ■ #012AB4,■ #103BB8, and ■#002CAD). These discrepancies are an inherent result of environmental variables, such as varying light intensity, shadows, and the specific color calibration of different IP camera sensors.

To maintain query robustness and uncalibrated environment, PERQS does not use strict string equality for color predicates. Instead, it employs a Fuzzy Evaluation Procedure integrated directly into the consensus engine. When the system receives a query for a specific color (e.g., vehicleColor="#0000FF"), the consensus policy decomposes the hex values into their constituent Red, Green, and Blue channels. As defined in the system's consensus rules (Table 3.1), a ±5% tolerance is applied to each RGB channel. Any detected color falling within this 5% Euclidean distance in the RGB color space is considered a logical "equal" for the purpose of consensus.

Figure 3.9: Case study 2: all the three cycles captured by camera 15 at 10:01:06.10.

Table 3.6: Case study 2: final result after time-travel consensus.

timeAppeared	vehicleType
10:00:14.20 10:01:13.10	cycle
10:01:02.30 10:01:45.40	cycle
10:01:04.50 10:01:42.30	cycle

Case Study 2 - Time-travel consensus: Find a cyclist passing through junction eight. PERQL query is as follows.

```
SELECT timeAppeared, vehicleType
FROM vehicleDetector
AT 42.525678, -90.723601
WHERE vehicleType="cycle"
TIME 2023-01-25 10:00:00.000 TO 2023-01-25 10:05:00.000
```

The execution process is similar to Case Study 1, except that junction eight lacks overlapping camera views. Only two of the six nearby cameras capture the junction's view. Therefore, we utilize time-travel consensus to merge the outcomes from the six cameras. One camera missed the cycles entirely because cycles took a turn before entering the field of view. The remaining cameras record the cycle, some a few seconds prior and another a few seconds after the junction. By merging the outcomes from these five cameras, we achieved a majority, hence the outcome. Steps 2 and 3 remain unchanged, but in step 4, we utilize time-travel consensus instead of a simple majority. The result is shown in Table 3.6.

Case Study 3 - Byzantine robustness: Find a white truck that passed through junction

IV at around 11:45 AM. Five cameras are available at this junction, each owned by a different organization. The corresponding PERQL query is shown below.

```
SELECT timeAppeared, vehicleType, vehicleColor, vehicleModel
FROM vehicleDetector
AT 42.556198, -90.734640
WHERE vehicleType="truck",
vehicleColor="#FFFFFF"
TIME 2023-01-25 11:45:00.000 TO 2023-01-25 11:50:00.000
```

This case study demonstrates the Byzantine robustness of PERQS in the presence of faulty and malicious participants.

Figure 3.10 shows representative frames captured by the five cameras covering junction IV. To simulate malicious behavior, we configured Organization 4, which owns Camera 19, to always report incorrect results. Specifically, it reports a *car* instead of a *truck* and alters the detected color information, intentionally producing misleading outputs to confuse the consensus algorithm. Additionally, we configured another camera to simulate a faulty participant by degrading its video quality. This camera has low resolution and inconsistent color reproduction due to poor lighting conditions, making it likely to report incomplete or incorrect vehicle attributes.

Despite the presence of one malicious camera and one faulty camera, the remaining three cameras correctly detected the white truck and its attributes. As a result, PERQS was able to reach a consensus based on the honest majority. This experiment confirms that the system can tolerate Byzantine behavior and unreliable camera inputs while still producing correct and trustworthy query results through majority-based consensus.

3.8.6 Baseline Discussions

In this section, we discuss how PERQS differs from traditional relational databases such as PostgreSQL [147] and DuckDB [126], as well as centralized video query systems such as FrameQL [77].

Comparison with Relational Databases: Traditional relational databases assume that analytics outputs are already materialized into structured tables. Queries such as searching for a “blue truck” can therefore execute efficiently over indexed metadata. However, these systems fundamentally assume centralized and trusted data ownership and do not support collaborative verification across multiple organizations.

PERQS addresses a different problem setting where raw video feeds remain distributed

Figure 3.10: Case Study 3: White truck captured by Cameras 16, 17, 18, 19, and 20 for the find white truck query.

across mutually untrusted participants. Instead of querying centrally stored data, PERQS focuses on generating collectively verified query results through distributed consensus.

Furthermore, conventional SQL systems do not natively support fuzzy reconciliation across heterogeneous analytics outputs. PERQS directly integrates configurable fuzzy consensus into query execution, including tolerance-based numerical matching, color similarity evaluation, temporal overlap reconciliation, and majority-based aggregation. Implementing similar functionality in PostgreSQL or DuckDB would require complex User Defined Functions (UDFs) and significant application-layer orchestration.

Comparison with Centralized Video Query Engines: Centralized video query systems primarily optimize query execution over large centralized video repositories where all data is already accessible within a trusted infrastructure. In contrast, PERQS is designed for distributed contributory surveillance networks where organizations retain ownership of their local video feeds and only share analytics outputs required for consensus formation.

Unlike centralized systems that treat detector outputs as ground truth, PERQS assumes that observations may be noisy, incomplete, or malicious. The system therefore incorporates consensus and integrity verification directly into the query lifecycle to improve robustness across unreliable participants.

Blind Spot Handling: Traditional databases and centralized video query systems generally return null results when direct observations are unavailable. PERQS instead introduces time-travel consensus, which reconstructs events indirectly using neighboring cameras and spatiotemporal correlation. This enables the system to recover observations in blind-spot scenarios

where single-camera approaches fail completely.

3.8.7 Optimizations

The main bottleneck that hinders fast query execution is the time-consuming process of video analysis. To improve performance, we have implemented two optimizations. The first one involves parallelizing the object detection process. Instead of processing frames one by one, we utilized the power of the GPU to process multiple frames simultaneously, which resulted in a significant improvement in the speed of video analysis. The 3-minute video object detection will be finished in less than 1 minute. We can improve performance further by fine-tuning the parameters and using more efficient models.

To further improve the efficiency of video analytics pipelines, video streams are divided into smaller chunks, typically five minutes or less in duration. Whenever a video chunk is finalized, the PERQS client computes a cryptographic hash of the chunk and submits it to the permissioned blockchain through a smart contract transaction.

Caching Optimization. In the second optimization we cache video analytics outputs after initial processing. Subsequent aggregate or filter queries can therefore execute directly on cached metadata rather than re-running expensive video analysis pipelines. This creates a significant warm-cache advantage. Cold-cache queries incur full analytics cost, while repeated queries over previously analyzed junctions execute near-instantaneously through metadata filtering alone. This optimization is particularly beneficial for recurring surveillance workloads involving aggregate statistics or repeated trajectory searches across the same junctions.

3.8.8 Artifact Availability

The PERQS detection pipeline and all scripts used for the evaluation will be made publicly available through the PERQS Git repository [73]. The repository includes the complete implementation for running vehicle detection, generating per-camera detection tables, and executing consensus. Users can download the CityFlowV2 dataset [35] and follow the provided workflow to reproduce the evaluation results presented in this thesis. A detailed README accompanies the repository, describing the software dependencies, execution steps, generation of intermediate output files, and the procedure for performing consensus over the generated detection tables. Together, the publicly available dataset and the released implementation enable researchers to extend the experimental results reported in this work.

3.9 Summary

PERQS is a contributory CCTV network framework that leverages blockchain technology for distributed integrity verification of video data. It enables multiple organizations to collaboratively share CCTV footage for public safety applications while preserving organizational data isolation.

Chapter 3 demonstrated how consensus across multiple video sources can improve robustness against unreliable or conflicting inputs, and how blockchain-based hash commitments can establish tamper-evident historical records for recorded footage. The evaluation showed that once video hashes are committed on-chain, subsequent modifications become detectable during verification.

Despite these advantages, PERQS has several limitations. In particular, it does not efficiently support joint video comparison operations across organizations. Further, while PERQS provides integrity guarantees for recorded data, the underlying permissioned blockchain infrastructure itself requires additional hardening. Addressing these challenges is the focus of Chapter 4.

Chapter 4

Aramid: Data Protection in Permissioned Blockchains

4.1 Introduction

In this chapter we present Aramid, which is an enhanced version of Hyperledger Fabric that offers data protection even in the presence of compromised blockchain components. We have used blockchain technology with PERQS for video hash verification. Blockchains guarantee integrity by tolerating a fraction of peer nodes being faulty or malicious based on the type of consensus protocol used. There is a large body of work on consensus algorithms providing different integrity guarantees [33, 118, 90, 86] and also work on verification and finding bugs in smart contracts [76, 102, 181] to ensure integrity. However, even a single misbehaving peer node (*e.g.,* due to compromise or an insider attack) could compromise data protection by leaking blockchain data to an attacker. Blockchains provide no protection against such attacks and they remain an unaddressed problem.

Blockchain data protection has primarily been addressed through application-layer mechanisms [130, 34, 122], such as encrypting the data at the application before storing them on the blockchain. This, however, negates the power of smart contracts to perform decentralized computation on the blockchain data. Techniques such as zero-knowledge proofs have been leveraged to provide additional application integrity guarantees while providing data confidentiality [25], such as proof against double spending or proof of solvency [45], *i.e.,* asset balances are non-negative at all points in time. Secure multi-party computation has been proposed for use with blockchains for more general purpose computation on private data [27]. These techniques still do not protect against a malicious peer revealing transaction logs or membership information, while also causing significant overhead in performance and application complexity.

We have used the permissioned blockchain platform Hyperledger Fabric [14] in PERQS implementations and evaluations. In Fabric, peer nodes that manage the distributed ledger could belong to different organizations. Peers can run smart contracts (called *chaincodes* in Fabric) that implement business logic and operate on the ledger data. Each peer can be part of one or more *channels*. Each channel is associated with a ledger and an independent consensus (or *ordering service*) to determine the order of transactions and blocks to be added to the ledger. The ledger stores transactions carried out within the channel. Fabric also maintains membership of organizations and peers in each channel. By the very nature of the application domains in which Fabric is used, both the ledger data and the membership of channels is deemed sensitive. That is, the identity of agents participating in one channel is not revealed to agents participating in another channel. The contents of each channel's ledger are also not revealed to other channels. This feature is useful in business settings in which a company can conduct business with two mutually-distrusting competitors. Such data partitioning allows implementation of anti-competition and data protection laws. Fabric also supports *private data collections* that allow certain data in a channel to only be shared among a subset of organizations. Organizations that are part of the private data collection may endorse, commit, and query the private data through smart contracts. Organizations that are not part of the collection only see a hash of the private data, that is stored in the transaction to ensure integrity. Private data collections are conceptually similar to channels, but operate via gossip messages exchanged between peers to provide a finer granularity of data privacy within a channel.

Unfortunately, Fabric (and similarly other permissioned blockchain implementations) provides a number of avenues by which confidentiality of channel data can be violated. Peers and other Fabric components run as containers. Leaks of confidential ledger data and cross-channel data leaks can happen whether through exploitable vulnerabilities in the containers, Fabric components, and smart contracts, or adversarial insiders seeking sabotage. We present a number of examples of such attacks in Section 4.4.

We build ***Aramid***, an enhanced version of Fabric that aims to improve data protection. The key problem in Fabric is that it offers too large an attack surface—implementing data protection atop Fabric would require placing trust on too many system components and business users. In Aramid, peers and other untrusted components that are part of multiple channels are partitioned to run as different instances, with a separate instance per channel. It ensures that data belonging to a channel is only accessible from system components and business users that have privileges to access that channel. Unlike Fabric, where blockchain components (*e.g.,* peers, orderers, smart contracts) must be trusted in order to achieve data protection, in Aramid, these components are untrusted. Instead, Aramid shifts the trust to vastly simpler and smaller *trusted*

proxies, as discussed below.

Partitioning peers into different instances by channel improves data protection, but breaks the existing Fabric abstraction of a single representative per enterprise (*e.g.,* in cases where a single peer from an enterprise participates in multiple channels). Aramid thus strives to preserve the existing abstractions in Fabric in an attempt to ensure that applications do not have to be rewritten. Aramid thus introduces trusted proxies that serve as the front-end to these privilege-separated peer instances. A client application interacting with an enterprise interacts transparently with the trusted proxy for that enterprise, which in turn routes the information to the suitable instance, based on channel information. Further, we use network policies to restrict communication of each Fabric component. For instance, each peer is permitted to communicate only with its trusted proxy and a smart contract process is only permitted to communicate with the peer that initiated it. All other attempted communication is dropped by the host network layer, protecting the system against malicious or sabotaged user (peer or smart contract) processes.

Aramid's privilege-separated architecture yields a vastly reduced attack surface in comparison to Fabric. In particular, peers and orderers are no longer part of Aramid's TCB (they are in Fabric), and are instead replaced by the significantly smaller and logically-simpler trusted proxy. The trusted proxy in Aramid is under 5% (by LOC of Go code) of the size of peers and orderers in Fabric (see Figure 4.3).

We have implemented Aramid for Fabric v2.2 and have deployed it atop the Kubernetes [89] container orchestration framework. Aramid uses the Istio service mesh [69] to enforce network-level policies for data protection. In our evaluation, we show that Aramid robustly defends against several data protection-violating attacks that are possible on Fabric, and that it does so with performance comparable to Fabric.

- **Identifying data protection failures in Fabric.** We argue that Fabric's data protection model requires implicitly trusting all system components in the permissioned blockchain. We present a number of attacks that are possible if this assumption is violated, *e.g.,* through buggy/compromised containers or insider attacks (Section 4.4).
- **Design and implementation of Aramid.** We present Aramid's privilege-separated architecture that improves data protection in the presence of compromised blockchain components (Section 4.5).
- **Evaluation.** We present a Robustness and performance evaluation that shows that the performance (throughput and latency) achieved at client applications that use Aramid is comparable to that of Fabric (Section 3.8).

4.2 Hyperledger Fabric

In this section, we provide background on Hyperledger Fabric [14] and its assumed threat model. We describe the different participant roles in Fabric along with some of the design constructs available to support data protection.

Peers are owned and operated by organizations and are the organization's connection points to the network. Each peer maintains a copy of the blockchain ledger and is responsible for its integrity. They execute business logic deployed as smart contracts (*a.k.a.* chaincodes) on them and each successful invocation of a smart contract is appended as a transaction to the blockchain ledger. Each peer is assigned a digital certificate by a certificate authority, which is used to sign and verify all communication. Smart contract execution (a transaction) is replicated across a subset of the peers, called *endorsers*, who compute the result of the execution and digitally sign it. Multiple concurrent transactions are ordered by *ordering service* nodes after which all peers verify and commit the transactions to the ledger. A peer process may connect to several other components over the network—client applications, ledger (a database), smart contract processes, ordering service nodes, and other peer nodes.

Channels are private sub-networks among member organizations (with peers), each having its own shared ledger, smart contracts, and ordering service nodes. A channel is a logical entity, which allows confidential communication and transactions among its members. Organizations that are not part of a channel should not have information about that channel. A single peer process belonging to an organization could participate in one or more channels and could thus maintain the ledgers of multiple channels using a single database instance. While a smart contract could be instantiated for use on multiple channels, each peer spawns a single process to run each smart contract. Thus, each smart contract process could execute transactions pertaining to one or more channels of which that peer is a member.

The ordering service (orderer nodes) create an order of transactions in a channel and bundle them into blocks. Orderers control access to the channel, restricting who can read and write to them and change its configuration. Client applications or users have identities on the blockchain (a digital certificate issued by a certificate authority) that may be distinct from the peer identities. They are external to the Fabric network and interact with the network via the Fabric SDK to install smart contracts on the network, create channels, submit transactions signed with their certificate that invoke smart contracts on a specific channel, and query the current state of the ledger. Figure 4.1 presents an example of a Fabric deployment.

In any organization participating in a blockchain platform, there are typically two personas of interest. The first is a *business user*, whose application data is recorded on the ledger through

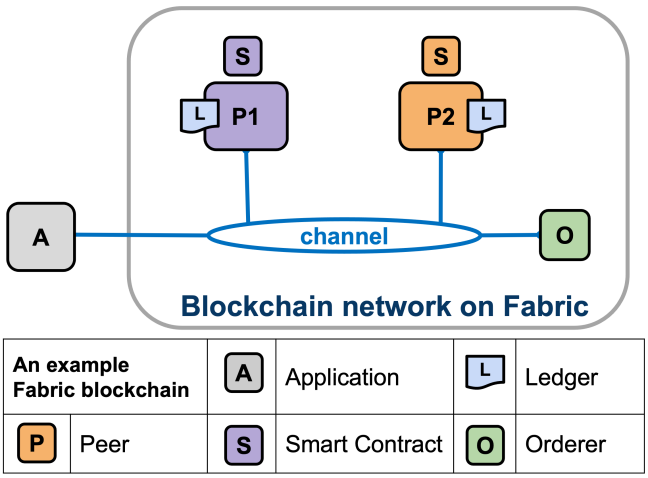

Figure 4.1: Fabric network with single channel and two peers.

transactions performed on the blockchain. They understand the value of the data and any confidentiality and protection needs associated with it. They may also be subject to compliance and audit of business and regulatory policies concerning data protection. The second is an *IT system administrator* who provides the infrastructure for hosting the blockchain, including the peer, ledger, smart contract processes and ordering service nodes. They are responsible for the availability and IT security of the infrastructure, but any data stored on the blockchain is only a string of bytes to them.

At this juncture, it is useful to understand the threat model of Fabric. Many other permissioned blockchain systems follow a similar threat model. Fabric supports different threat models for data integrity and confidentiality. It guarantees data integrity even when one or more peers (or an IT administrator controlling the peer process) are faulty or malicious, depending on the threat model supported by the consensus algorithm used. For instance, when using a crash fault-tolerant consensus algorithm such as Raft [117], it supports up to half the peers failing, but does not tolerate malicious peers. When using a byzantine fault tolerant consensus algorithm [32], it supports up to a third of the peer nodes being malicious and byzantine. In contrast, Fabric guarantees data confidentiality only when every peer (and every business user with access to data on the blockchain) is non-malicious and is operating as designed. The goal of our work is to extend Fabric's confidentiality guarantees even to situations when a Fabric peer, smart contract or ordering service process has been compromised by a faulty or malicious business user, without modifying any of the abstractions supported by Fabric. An compromised process or malicious business user will be unable to exfiltrate business information.

Specifically, Fabric's peer and smart contract processes can be compromised and leak data

to other processes or send unencrypted confidential data, which can be captured by an attacker. Further, a compromised peer process can leak secret channel-specific blockchain data and information to another Fabric process that is not part of that channel. Similarly, a compromised ordering service node can send blocks of a particular channel to peers who are not part of that channel.

4.3 Threat Model for Aramid

Aramid aims to prevent attacks by compromised or malicious Fabric system components that violate data protection. Aramid relieves blockchain business users who control all Fabric processes and the ledger in an organization from the obligation of data protection. We consider that *all Fabric processes and business users that can access them are untrusted*, and can be compromised and act maliciously. Aramid provides systemic means for enforcing data protection rules at the infrastructure layer, following the principle of privilege separation. This architecture offers a vastly reduced attack surface in comparison to Fabric.

On average a single IT administrator may manage tens or even hundreds of applications (and therefore hundreds or more business users using those applications). Further, they do not comprehend the value of the bytes stored to maliciously leak them. By *trusting only the infrastructure layer and few IT administrators who manage them*, Aramid also reduces the number of insiders trusted for data protection. Specifically, in the infrastructure layer, we trust the operating system for not modifying and leaking packet information. We run all processes in a Kubernetes [89]-based infrastructure, and we trust the Kubernetes control plane for providing pod-level isolation (a *pod* is a group of one or more containers in Kubernetes), network, and storage security. We trust Istio [69] to enforce authorization policies and network rules. We note that Kubernetes and Istio are aspects of our specific implementation and can be readily replaced with other container orchestrators and network policy frameworks.

A malicious insider (business user) on Fabric can access all data from all channels in which the business participates. While Aramid reduces the attack surface, it does not eliminate data exposure. For instance, a malicious business insider with credentials to access a particular channel's data can still leak that channel's data via out-of-band methods (*i.e.,* out of the purview of the blockchain). Future extensions could combine Aramid with prior art (*e.g.,* [104]) to mitigate such attacks.

Attacks outside this scope—including freshness attacks, consensus-level attacks, denial-of-service attacks, and cryptographic breaks—are considered out of scope and are not addressed in this work.

4.3.1 Comparison with Permissionless Blockchains

Permissionless blockchains such as Bitcoin rely on an honest majority assumption among unknown and potentially adversarial participants. Any party can join the network, which significantly increases the attack surface and diversity of adversarial behaviors. Security in such systems is typically derived from economic incentives and cryptographic consensus mechanisms.

In contrast, Aramid operates in a permissioned blockchain environment, such as Hyperledger Fabric, where participants are known and authenticated. However, Aramid assumes that participants' software instances are untrusted, even though their identities are known. Participants may attempt to deviate from the protocol to gain competitive advantage, access private data beyond their authorization, or increase profit.

Thus, unlike traditional permissioned systems that assume organizational trust, Aramid adopts a zero-trust posture toward business-level components. The key difference lies in the trust boundary placement:

- In permissionless blockchains: Trust is distributed across the entire protocol and enforced through consensus among anonymous participants.
- In traditional permissioned blockchains: Trust is placed in known organizational peers.
- In Aramid: Trust is shifted downward to a minimal, infrastructure-level enforcement layer.

A primary contribution of Aramid is reducing the Trusted Computing Base (TCB). In standard Fabric deployments, the entire peer implementation must behave correctly to ensure data isolation. Aramid replaces this broad trust assumption with a small, auditable proxy responsible solely for enforcing isolation.

In permissionless systems, the full protocol stack effectively forms the TCB. In Aramid, trust is confined to a minimal infrastructure component, improving auditability and reducing systemic risk.

4.4 Motivating Examples

To motivate examples of data protection violations in Fabric, we consider an example business scenario and present several attacks that are possible with Fabric. We emphasize that these attacks are merely illustrative, and not in any way a comprehensive listing of data protection attacks that Aramid defends against.

Consider a Fabric deployment that is used by shipping companies and delivery companies (*e.g.,* TradeLens [148]). Shipping companies publish a list of available container slots on their ships. Delivery companies bid for these container slots, with each delivery company submitting a bid that consists of the number of slots it is bidding for, and the price offered per slot. Each shipping company then allocates container slots based on a variety of business considerations, such as the price offered per slot, long-term relationship with the delivery company, its reputation, and prior business history with that company.

Each shipping company's algorithm to select bid winners is proprietary, and the shipping company may also wish to keep secret the list of bids that it receives. Similarly, each delivery company will want to protect the confidentiality of the price that it quotes and the number of slots that it requests. In Fabric, channels and private data collections serve as the core mechanism for data protection, allowing these goals to be met in a deployment that includes all shipping and delivery companies as authorized participants. For example, a Fabric deployment could contain:

- A single public channel, available to all shipping and delivery companies, to share common data among all participants. This channel's ledger records the availability of container slots, shipping data, and other common data.
- One channel per pair of shipping and delivery companies that wish to do business with each other. The channel's ledger records data private to that pair of companies, *e.g.,* the bid submitted by the delivery company to that shipping company.
- One channel per shipping company and consortium of delivery companies that wish to collaborate in submitting a bid. For example, an individual delivery company may not have sufficiently many goods to fill a single container, in which case it may choose to share (and submit a bid for) a container together with collaborating delivery companies. The ledger of this channel records joint bids submitted by the consortium of delivery companies.
- One private data collection per shipping company that stores the cost of available container slots. The private data collection ensures that the cost of an available slot is known only to that shipping company; however, availability information of the slots is known to all channel members on the public channel.

The permission infrastructure of Fabric ensures that only channel members can access the ledger of that channel. In fact, Fabric's infrastructure even protects channel-membership information, *i.e.,* it keeps confidential the fact that a participant is a member of a particular channel. Thus, for instance, it would not be possible in the above example for a business user in one delivery company to determine if its competitor is in business with a particular ship-

ping company. As another example, one member of a consortium of delivery companies cannot determine if another member is part of a second consortium.

While these security checks are embedded in Fabric's permission-checking mechanism, they offer data protection only under the implicit assumption that all components of the blockchain deployment are well-behaved. Unfortunately, as multiple CVEs and bug-reports show (*e.g.,* [158, 161, 159, 119, 160, 4, 1, 5, 2, 3, 136, 61]), this assumption can easily be broken by compromised blockchain components (*e.g.,* peers, orderers) and malicious business users.

We present below several examples of attacks that violate data protection in the above setting:

① **Sending data to entities outside the blockchain.** Fabric's permission infrastructure protects a channel's ledger data by ensuring that only peers with channel membership can access it. However, Fabric does not have in-built mechanisms that prevent a malicious peer from leaking ledger data to entities outside the blockchain. In the above example, a malicious peer belonging to a shipping company can leak the bids that it receives to a public website, thereby compromising the private information included in the bids.

② **Sending unencrypted data.** A compromised or malicious peer can send data unencrypted. A network-based adversary snooping on network links can read all the data sent by the peer. This may include sensitive data, transaction inputs and outputs processed by smart contracts that the peer executes. This attack has the same effect as that of sending data to entities outside the blockchain.

③ **Intra-container data leaks.** Fabric components run within Docker containers on a Kubernetes cluster. In a maliciously-configured container, a Fabric-related process (*e.g.,* one running the code of a peer) could leak data to other processes within that container, which in turn may send data to external unauthorized entities.

④ **Inter-channel data leaks.** Fabric relies on channels as the fundamental data partitioning abstraction. However, a malicious Fabric component can leak data that is accessible to it. For example, a compromised peer node belonging to a delivery company that is a member of multiple consortia (each with its own channel) can leak the data stored in the ledger of one consortium to the other consortia of which it is a member. This is because, on Fabric, the *same* peer node is part of both channels. Thus, data from both channels is visible to the peer node, and a malicious peer node can therefore serve as a vehicle for an inter-channel data leak. Similar inter-channel leaks are possible via buggy (or malicious) smart contracts running on the peer node, or a malicious ordering service, which can view all the transactions being appended to the ledger.

⑤ Leaking channel membership information. In Fabric, even membership in a channel is protected information that is privy only to members of that channel. In the example above, a compromised peer belonging to a delivery company that is part of multiple consortia can leak information about the members of one consortium, either to other consortia or to external entities.

⑥ Leaking smart contract business logic information. The business logic within smart contracts (*e.g.,* the proprietary algorithm used by a shipping company for bid selection) is considered confidential. The code of a smart contract executing in one channel is only known to members of the channel, in fact, only known to the endorsers for that smart contract. A malicious system component process could leak business logic information to outside the channel.

⑦ Leaks via non-channel messages. In Fabric, peer nodes send out messages that are not tied to any individual channels. For example, peer nodes send out liveness messages (`isAlive`) or `Ping` messages. A malicious peer can leak channel data to an adversary by encoding sensitive information as part of these non-channel messages.

⑧ Leaking data stored in private data collections. Fabric supports the notion of private data collections. It allows a subset of channel members to form a private collection and transact privately. A ledger is not maintained for a private data collection; rather, peers that are part of the collection interact by exchanging gossip messages. Private data in the collection is accessible to only the collection members defined by the collection policy. However, a malicious peer can easily leak private data accessible to it to a non-collection member. In the above example, a malicious peer of a shipping company can leak the container slot cost to a delivery company. The delivery company can then choose its bid not more than the learned container slot cost.

To summarize, Fabric provides the channel abstraction (and the related notion of private data collections) for data protection. The mechanisms to enforce this protection are part of Fabric's permission model, but Fabric assumes that the various system components are not malicious. In the presence of malicious components, such as peers, smart contracts and ordering services, the assumptions that Fabric makes no longer hold, leading to the kinds of attacks discussed above. Aramid aims to improve Fabric by providing data protection even in the presence of malicious components.

4.5 Aramid

Aramid extends Fabric's data security guarantees even to situations when Fabric system components are compromised. We designed Aramid with three objectives in mind:

• **Reduce the number and complexity of trusted components:** As already discussed, Fabric's channel-based data protection mechanisms only work as expected when each component (peers, orderers, smart contracts) behave honestly. In a large blockchain deployment this assumption may no longer hold. Aramid aims to offer data protection while trusting far fewer and logically-simpler components than on Fabric.

• **Multi-faceted policy enforcement:** Aramid enforces policies both at the network layer, ensuring messages are visible only to (authorized) participants in the blockchain, as well as Fabric-specific policies. For instance, the notion of channels is unique to Fabric, and Aramid ensures that channel messages are visible only to other nodes that are part of the channel.

• **Preserve Fabric components and abstractions:** Aramid is transparent to legacy Fabric applications. Existing applications do not have to reprogrammed. Thus, any interaction with a peer node in Fabric will continue in the same way in Aramid. Aramid transparently introduces new, trusted network elements that enforce policies without breaking the illusion of communicating directly with a peer node. Moreover, Aramid can operate and provide a measure of protection even if its mechanisms are adopted by only *some* participating entities of the blockchain network. Note that Aramid will offer weaker data protection in such partial deployments (*e.g.,* some data leakage channels may still exist), but even a partial deployment (*i.e.,* by a subset of participating entities) improves upon the state-of-the-art as offered by Fabric.

4.5.1 Enforcing Channel Data Separation

Fabric's design allows a peer node (*i.e.,* a process running the peer code in a container instance) to be part of multiple channels, and consequently, access the ledgers that are private to these channels. As shown in Section 4.4, a compromised or malicious peer process can leak sensitive channel data to any network member who does not have explicit access to that channel's data.

Aramid's core design is based on privilege separation that replaces a single peer node with multiple peer nodes such that each peer is part of only one channel and maintains only one ledger (illustrated in Figure 4.2). Each such peer node runs in a Docker container inside a fresh pod instance. Each peer node has its own smart contract instances for its channel as separate pods that it alone communicates with.

Simply disaggregating a single peer node in Fabric into multiple peers breaks transparency to applications, *e.g.,* other Fabric nodes and client nodes that interact with the peer to perform blockchain transactions. These applications will now have to route messages to different peer instances, based upon the channel on which the corresponding messages are being exchanged. Unfortunately, this will break existing applications. To maintain transparency, Aramid instead

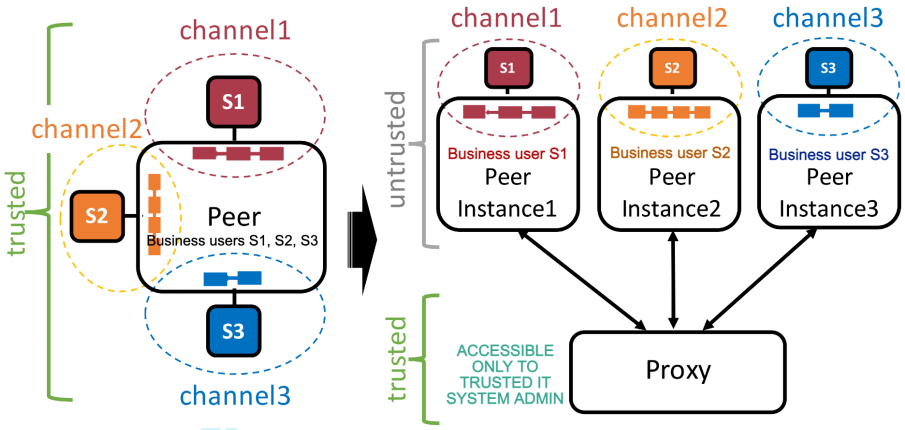

Figure 4.2: This figure compares how Fabric and Aramid organize peers. A single peer instance that is part of multiple channels in Fabric is replaced by multiple peer instances (one per channel) in Aramid. Each peer can run smart contracts (denoted by "S1", "S2", "S3") and can only access the ledger of that channel. A trusted proxy acts as a front for the multiple peer instances to external applications. The proxy inspects messages directed to the peers, and routes the messages appropriately, based on the channel information. In Aramid, peers are untrusted, and the trust is instead shifted to a proxy node.

introduces a *trusted proxy node* for the peer instances. Applications that interacted with the single peer in Fabric interact instead with this trusted proxy in Aramid, which then routes messages to the appropriate peer instance based on channel information. The proxy node is part of Aramid's TCB and transparently mediates all communications to and from these peer instances. The collection of proxy and peer instances represent a single "peer" entity from the perspective of the Fabric network as well as any client applications interacting with it.

Aramid uses the same design for orderer nodes as well. In Aramid, the orderer node is disaggregated so that each channel has its own orderer node that runs in a Docker container inside a fresh pod instance. A trusted proxy then serves as the front-end to the collection of orderer nodes, and routes all communication to the appropriate orderer node based on the channel information. As before, the collection of proxy and orderer instances represents a single "orderer" entity from the perspective of legacy Fabric applications.

Aramid enforces two kinds of policies:

① *Fabric-specific policies* that are enforced by trusted proxies. The most notable of these policies is that two communicating Fabric components (*e.g.,* peer instances) that exchange channel-specific messages must belong to the same channel. As we will describe, the trusted proxy has the privilege to inspect all incoming messages to and outgoing messages from the peers for which it serves as a proxy. If the above policy is violated, then the proxy silently

Component	LOC (Go code)	# Vendor modules used
Peer	1,41,031	18
Orderer	55,501	16
Trusted proxy	8,541	13

Figure 4.3: Sizes of various components in Fabric (shaded gray) and Aramid. Peers and orderers are part of Fabric's TCB. The trusted proxy is part of Aramid's TCB, but peers and orderers are not.

discards the message. Although we have illustrated proxies so far only for peer nodes, such trusted proxies can be added for every Fabric component. For example, an orderer proxy would verify that blocks are sent only to channel members based on the channel information present in the block header and the channel that an orderer node serves (each orderer node instance serves only one channel to ensure privilege separation). The orderer itself is untrusted in Aramid.

② *Generic network policies* that apply to any permissioned blockchain deployment. These policies are enforced by the underlying service mesh and container orchestration framework (Istio and Kubernetes, in our implementation). The trusted proxy interacts with the service mesh and dynamically adds these network policies that restrict communication paths. For example, these policies ensure that a peer instance can only communicate with its proxy node, hosted ledger, and smart contracts. They also restrict a smart contract and ledger instance to connect only to the peer instance that has hosted them, and ensure that all network communication is encrypted using TLS.

At first blush, it may appear that Aramid's design simply shifts the trust boundary from peer and orderer nodes to the trusted proxy nodes. While this is indeed the case, there are two key benefits to this approach:

① System components such as peers and orderers are complex pieces of code that have evolved over time, and contain complex logic to handle the plethora of messages and corner cases that Fabric supports. In contrast, the trusted proxy is extremely simple in its functionality, and only serves to route messages to the correct peer or orderer node, based on channel information. Because messages on the Fabric network are encrypted, the trusted proxy also manages TLS keys on behalf of the peers and orderers so that it can inspect the packets for channel information (details in Section 4.5.2). The proxy implements no other functionality besides routing, key-management, and packet header inspection to glean channel information.

② The trusted proxy is managed by IT system administrators, and is not accessible to business users. Aramid does not trust business users, who are cognizant of the value of the ledger data.

As a result of this simplicity, the attack surface of the trusted proxy in Aramid is vastly

smaller than that of peer nodes and orderer nodes, which are part of the TCB in Fabric. We quantify this reduction in Figure 4.3, which shows the sizes (in lines of Go code) of the components that are part of the TCB in Fabric versus the size of the trusted component (the core of Aramid's TCB). We also show the number of vendor modules used by each component.[1] In the current prototype of Aramid, we used vendor modules that are already part of Fabric's code-base to implement the proxy. These modules offer various functionalities as implemented in standard libraries. It would be possible to further debloat the TCB in Aramid by only retaining the functionality required by the trusted proxy and stripping away the rest; however, we have not attempted this exercise.

4.5.2 Implementation

We have implemented a prototype of Aramid for Fabric v2.2 and deployed it on Kubernetes [89], a popular container management platform for deploying and managing Fabric networks. Each Fabric component, including peer instances, smart contracts, ledger, orderer, and the trusted proxy, runs in a separate Kubernetes pod. We leverage the open-source service mesh Istio [69] to ensure that only encrypted messages are exchanged on the network. Istio deploys a sidecar proxy in every pod that intercepts all the communications between pods and provides transparent TLS encryption. Our proxies run with the privileges that allow them to interact with the Kubernetes' control plane. Proxies can also dynamically add or remove network rules to the service mesh. The network rules specify which network paths are allowed and are enforced by the Kubernetes platform. The proxy uses the Kubernetes Golang API [88] to create pods and add policies in the cluster dynamically. Listing 4.1 shows an example of network policy for a peer instance, allowing it to connect only the trusted proxy and smart contract pods on the appropriate ports.

To an end-user or a client application, Aramid's architecture appears similar to that of Fabric. End-users interact with blockchain network components via a client application. The client application (henceforth, client) registers and enrolls with the organization's (*i.e.,* the member organization participating in the permissioned blockchain) certificate authority (CA) and receives the necessary cryptographic material. The client uses this cryptographic identity to authenticate itself to the network. Aramid ensures that the client is unaware of the existence of a proxy and multiple peer instances or multiple orderer instances, one per channel. The client interacts with the network using the same commands as in Fabric.

We next describe in detail how different transaction flows and design aspects of Fabric are

[1] Vendor modules are akin to libraries that implement standard functionality. We did not count the lines of code in these vendor modules, but rather chose to count the number of modules used.

Listing 4.1: Example of a network policy (enforced by the service mesh) for a peer instance to connect to its proxy.

```
# Example Policy for 'peer0-org1-instance1'
# Ingress policy only allows TCP connections on port 7051 from peer0-org1-proxy
# Egress policy also only allows connections on port 7051 to peer0-org1-proxy
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: peer0-org1-instance1-network-policy1
  namespace: hyperledger
spec:
  podSelector:
    matchLabels:
      app: peer0-org1-instance1
  policyTypes:
  - Ingress
  - Egress
  ingress:
  - from:
    - podSelector:
        matchLabels:
          app: peer0-org1-proxy
    ports:
    - protocol: TCP
      port: 7051
  egress:
  - to:
    - podSelector:
        matchLabels:
          app: peer0-org1-proxy
    ports:
    - protocol: TCP
      port: 7051
```


transparently supported to client applications in Aramid.

Peer identity and certificates: Each peer instance and the trusted proxy have their own identities, registered with the CA and the organization's membership service provider (MSP). The MSP is a Fabric component that abstracts away the cryptographic mechanisms and protocols involved with issuing certificates, validating certificates, and user authentication. The proxy runs with the identity of the vanilla Fabric peer. It generates new cryptographic material for each peer instance. This ensures that a peer instance cannot decrypt the traffic meant for other peers.

All communication in Fabric is encrypted end-to-end, *i.e.,* the communication from the client to the peer node happens over TLS. To preserve Fabric's abstractions in Aramid, each peer instance and the proxy are given their own TLS keys. For a client-to-peer connection in Aramid, the client forms a TLS connection to the proxy, and the proxy connects the client to the appropriate peer instance by forming a TLS connection to the peer. Hence, the proxy decrypts the gRPC stream from the client application with its TLS keys, inspects its contents,

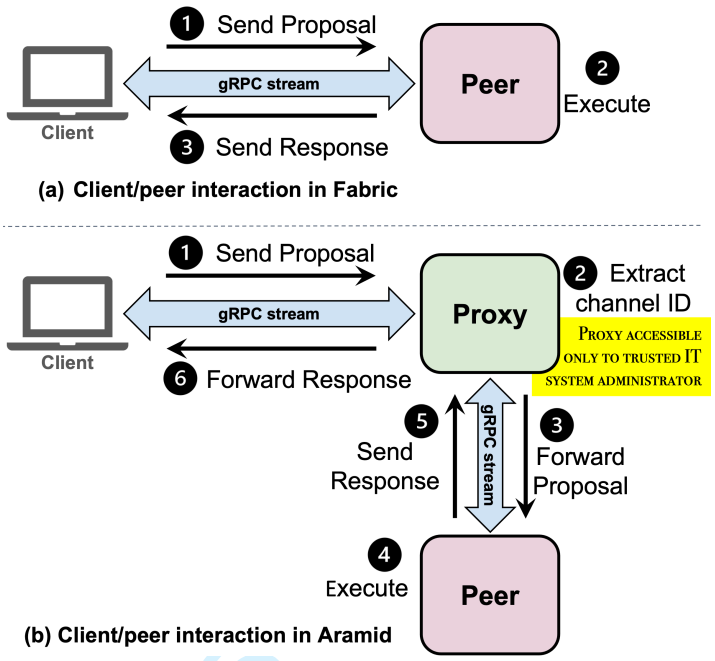

Figure 4.4: Client/peer interaction in (a) Fabric; and (b) and Aramid.

and reencrypts it for onward transmission to each peer instance. Further, each peer instance has its own signing key, which it uses to sign the messages and transaction endorsements.

Transaction submission: Figure 4.4(a) illustrates the interaction between a client and a peer in Fabric for transaction submission. It involves the following steps:

① *Send proposal.* The client initiates the transaction by creating a transaction proposal that includes the smart contract ID, function name, arguments, channel ID, and client certificate, among others. The Fabric SDK suitably formats and packages the transaction proposal (as a protocol buffer over gRPC) and takes the client's cryptographic credentials to produce a unique signature for this transaction proposal. This transaction proposal is then sent to one or more endorsing peers.

② *Execute transaction.* The peer performs verification checks on the transaction proposal. It then simulates the proposal by executing the operation on the specified smart contract. It prepares a transaction response with the result of the simulation.

③ *Send response.* The endorsing peer replies with the proposal response and its signature over the proposal response (called the *endorsement*).

In Aramid, the client-peer interaction is mediated by a trusted proxy, as shown in Figure 4.4(b). It provides the illusion to the client that it is interacting directly with the peer, as illustrated in the steps below:

① *Send proposal.* The client application initiates the transaction and creates a transaction proposal. It sets up a gRPC connection with the proxy. The client, under the illusion that the proxy is the endorsing peer, makes a gRPC function call to the proxy with the signed proposal.

② *Extract channel ID.* In the gRPC function, the proxy verifies the client's identity and extracts the channel ID from the content of the transaction proposal.

③ *Forward proposal.* The proxy keeps a record of the specific channel for which each peer instance is responsible. Using this mapping information and channel ID (extracted in step 2), it finds the correct peer instance. It then makes the gRPC function call with the signed transaction proposal on behalf of the client and waits for the response.

④ *Execute transaction.* The peer instance performs verification checks on the transaction proposal. It then executes the transaction on the specified smart contract and produces the response. This step is identical to Fabric.

⑤ *Send response.* The peer instance returns the signed transaction response to the proxy. The peer instance code is identical to a Fabric peer instance (*i.e.,* unmodified).

⑥ *Forward response.* The proxy returns the gRPC function call with the signed response from the peer instance.

Transaction ordering and commit: Note that in Aramid, the transaction flow remains identical to Fabric from the client's perspective (illustrated in Figure 4.5). In Fabric, the client application collects endorsements from the endorsing organizations, as discussed above. After receiving sufficiently-many endorsements, it sends the transaction proposal and response within a transaction message to the ordering service. The ordering service creates blocks of transactions for each channel. Aramid runs one of the proxies as an *anchor peer* for the organization. An anchor peer is a designated peer in an organization which gets blocks from the orderer. As a result, the ordering service sends all the blocks to the anchor peer proxy, which then disseminates the blocks to its peer instances and other proxies. Each proxy inspects the block message and extracts the channel ID. It finds the peer instance for that channel; if the channel does not exist (it doesn't belong to the channel), it discards the block message, else it forwards the block message to the peer instance. The peer instance performs verification checks on the block message. It then validates the transactions in the block and updates the ledger. Finally, the block is appended to the blockchain (*i.e.,* transaction commit).

Block event listener: In Fabric, after submitting a transaction, the client application uses the Fabric SDK to register a *listener* to receive blocks as they are added to the channel ledger. Fabric SDK processes the incoming blocks and looks for submitted transaction IDs. If the transaction is found in the block, it notifies the client about the transaction status (*i.e.,* whether

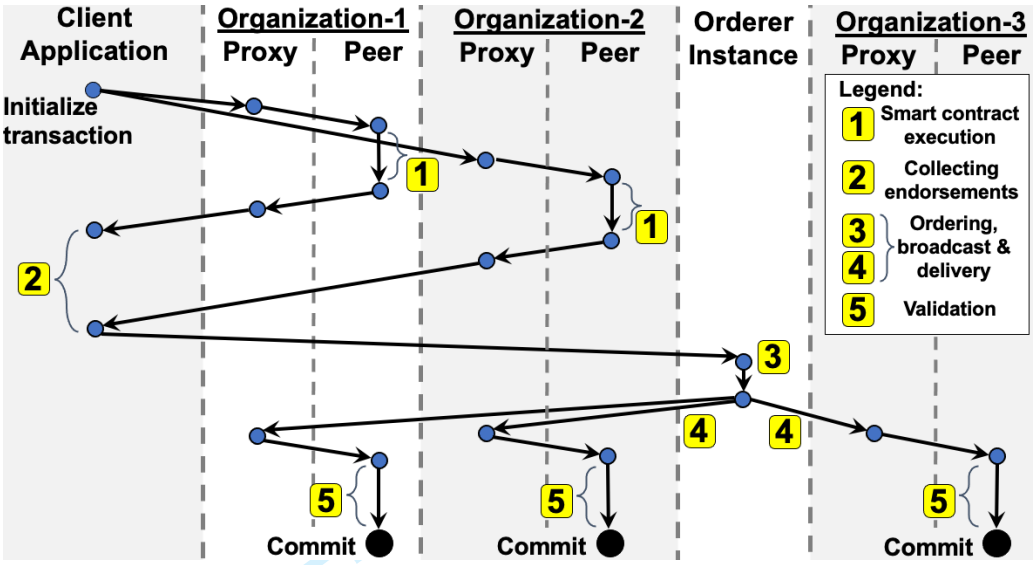

Figure 4.5: Interaction between components in Aramid in fulfilling a transaction. In this example, a transaction initiated by the client executes a smart contract at Organization-1 and Organization-2, following which the endorsements are collected, and the endorsed transactions are sent to the ordering service. The orderer then broadcasts it to all the peers, who validate the transaction and add it to their ledgers.

the transaction completed successfully or not). In Aramid, the client registers a listener with the proxy, which in turn registers a listener with the peer on behalf of the client. Whenever the peer adds a block to the ledger, it sends the block to the proxy's listener, and the proxy sends it back to the client listener. The client-side then processes the block and looks for submitted transaction IDs. It marks a transaction as completed only if the transaction is found in a block before the timeout occurs.

Request for peer to join a channel: The proxy first checks that the channel does not already exist. It then creates a peer instance and forwards the join channel request to that instance. It waits for the peer's response. If the channel join is successful, the proxy sends a successful response to the client application and stores the mapping between channel ID and the peer instance pod. It adds Kubernetes network policies to restrict the peer instance to communicate only to the proxy. Since pod creation and initialization can take some time, we create a small pool of peer instances beforehand and map them to channel ID on request. A peer leaving a channel is handled similarly and we do not elaborate on that in the interest of brevity.

Smart contract (chaincode) install and commit: Fabric v2.x adds support for smart contracts to be run outside the peer container. We leverage this capability in our implementation. The smart contract is compiled and installed in a separate pod instead of within the peer pod; the

only code installed in the peer process is the information required to connect to the external smart contract process. This ensures that the peer pod does not have access to the smart contract logic. To achieve this goal, a smart contract package is created with information on connecting to the smart contract server, including its address, TLS certificates, and dial timeout configurations. Smart contract installation involves three steps:

① *Install.* When the proxy receives the smart contract installation request on a channel, it creates a smart contract pod and smart contract package with the connection information. Since the smart contract install request does not mention the channel information, the proxy sends the smart contract package to all the peer instances. The peers install the smart contract package using the smart contract `lifecycle` process and send back the package identifier. The proxy forwards it to the client application.

② *Approve.* Once the install step is done, the client sends a request to approve the smart contract. The approve request contains the channel information. The proxy looks for the peer instance running on that channel and sends the approve request to it. The proxy gets the approval from the peer instance and relays it back to the client. At this time, the proxy adds network policies to enable traffic between the smart contract pod and the peer instance.

③ *Commit.* Once sufficiently many organizations on the channel have installed and approved the smart contract, any client can commit the smart contract definition on the channel. To commit, the client first needs to collect endorsements from the organizations that have approved the smart contract. We have discussed how a client collects the endorsements in our design. The client then submits the transaction to the ordering service and waits for the block. The ordering service adds the commit transaction to the block and delivers the block to the proxies on the channel. Proxies deliver the block to the peer instances. Peers process the block and add it to the ledger; the block is then sent to the client via the proxy.

If the transaction is successful, the smart contract's installation is complete and it can be invoked.

Persistent data storage: Storage volumes are mounted to peer instances by the proxy at the time of pod creation. These persistent storage volumes are used to store committed blocks. Apart from the persistent storage volume, all other data, including certificates and signing keys, are mounted as read-only volumes. The proxy ensures that the persistent storage volumes are mounted in ReadWriteOnce mode so that the same volume cannot be mounted by a malicious peer or any other pod in the system, thus preventing data leakage through the file system.

Ledger database options in Fabric: Fabric supports two alternatives for the ledger database, an embedded database using Goleveldb or a CouchDB [15] database. Our current implementation

of Aramid is based on Goleveldb, but the implementation can be easily extended to support CouchDB based on the principles already discussed. When a peer joins a channel, besides creating the peer pod instance, a CouchDB instance is also created along with the necessary network policies to ensure that the CouchDB instance communicates only with the specific peer instance for that channel. This ensures that the ledgers of different channels are isolated.

Handling channel-agnostic messages: As discussed in this section, Aramid's proxy takes routing decisions based on the channel information present in the messages. However, Fabric has several types of messages that do not contain channel information. Aramid needs to handle these messages differently depending on the message type and the action required by the message. For instance, we discussed how we handle the message request to install a smart contract, which does not specify the channel. Other types of non-channel messages are based on gossip protocol and are used to indicate liveness, such as `isAlive` and `ping`. In Fabric, the code of the peer contains the logic to respond to these messages. In keeping with Aramid's design philosophy of reusing the code of Fabric components where possible, the code components to respond to these channel agnostic messages is reused in the proxy.

When the proxy receives a non-channel gossip message (say `isAlive`) from another peer, the proxy simply responds to the sender based on the peer pod status and does not forward it to any of its peer instances.

Private data collections: Fabric supports the notion of private data collections, where a subset of organizations within a channel can endorse, commit and query private data through smart contracts without having to create a separate channel. The private data is shared among the subset of authorized organizations using gossip messages and even the ordering service does not obtain a copy of the private data as part of the transactions. Only a hash of the private data is included in the transaction. This is an additional attack vector wherein a malicious peer instance can leak private data to unauthorized recipients within or outside the channel.

Aramid supports private data collections and also prevents the leak of private data by a malicious peer. The proxy joins all the channels for which it has peer instances; however, it does not maintain the ledger. These peer instances can directly communicate with the proxy using gossip messages. When a peer instance executes a private data transaction, it checks its collection policy and disseminates the private data to other eligible collection members, *viz.* the proxy in our case. The proxy upon reception of the private data checks the collection policy. It then sends the private data to the authorized organization proxies, which forward this data to the peer instances running on that channel. Thus, we check the collection policy at the source proxy and ensure that the private data is not sent to an organization outside that collection.

Peer instances do not need to create any outbound connections except with their proxy

and the smart contract pod. Aramid enforces this by setting the network rules suitably in the underlying Kubernetes cluster. Since network rules are defined using pod labels, Kubernetes infrastructure will ensure that the policy is followed even after a dynamic pod recreation (*e.g.,* when a pod crashes and is automatically restarted by Kubernetes). The proxy reinitiates the connections to instances on a pod restart.

4.6 Evaluation

In this evaluation, we focus on two objectives: (1) qualitatively demonstrating robustness against representative cross-channel data leakage attacks, and (2) quantifying the performance overheads introduced by Aramid's privilege-separation architecture in terms of latency and throughput.

Our evaluation specifically targets infrastructure-level isolation and the impact of the trusted proxy on transaction processing. We do not provide a formal security proof or an exhaustive analysis of all possible attacks, such as consensus-level failures or cryptographic breaks, which are outside the scope of this work. Instead, the evaluation verifies that Aramid effectively prevents the common data protection violations observed in standard Hyperledger Fabric deployments.

We implemented a Fabric-based testbed consisting of three organizations, with one orderer node and one peer node per organization. The default configuration parameters used for the peer and orderer nodes are summarized in Figure 4.6.

4.6.0.1 Setup for fabric testbed

For the performance evaluation experiments, we use Hyperledger Caliper [65] as our workload generator. We run all of our experiments with the Marbles smart contract [29]. We run each organization on a separate Kubernetes cluster node and the resource configuration is kept same for a head-to-head performance comparison of Fabric and Aramid. Our experimental network configuration matches those in real deployments, so we believe that the benchmark results are representative of a real-world deployment of Aramid.

4.6.1 Evaluating Aramid's Robustness

In Section 4.4, we listed the attacks that can be launched in Fabric. We now show how Aramid successfully defends against these attacks, focusing on the mechanism(s) that prevents each of these attacks. We first review the mechanisms in Aramid:

• **TLS by default.** Aramid uses the open-source service mesh Istio [69] to ensure that all the data leaving a pod is encrypted. Istio deploys a sidecar proxy in every pod that intercepts

Parameter	Values
No of Channels :	2
State Database :	GoLevelDB
Endorsement Policy :	Any one peer on channel endorses
Batch Size :	500 transactions per block
Batch Timeout :	2 sec
Validation Pool Size :	64
Caliper Transaction Send Rate :	1500 tps
Caliper Workers :	16 (local)
Smart Contract (Chaincode) :	Marbles [29]
Chaincode Transaction :	initMarble [29]

Figure 4.6: Default configuration used for experiments.

Name	Parameters	Pods
Kmaster	8vCPUs, 32GB RAM	Master node
kworker-0	24vCPUs, 64GB RAM	Caliper pods
kworker-1	16vCPUs, 32GB RAM	Organization-1 pods
kworker-2	16vCPUs, 32GB RAM	Organization-2 pods
kworker-3	16vCPUs, 32GB RAM	Organization-3 pods
kworker-4	8vCPUs, 32GB RAM	Orderer pods

Figure 4.7: Kubernetes cluster setup used in our experiments. The "Pods" column shows the main pods that we run in each cluster node.

all communications between pods. It provides transparent TLS encryption support without requiring any application changes.

• **Trusted proxy.** The trusted proxy provides complete mediation for all communications to and from Fabric components (such as peers and orderers) and enforces Fabric-specific policies. It makes routing decisions based on message content and network information.

• **Network policies.** Kubernetes network policies are firewall rules which monitor traffic to, from, and between pods. Aramid leverages these rules to block all traffic that is not explicitly allowed. Trusted proxies in Aramid define these network rules dynamically in the form of network paths for each pod, when it is created.

We now discuss which component of Aramid prevents the various attacks listed in Section 4.4.

① **Sending data to entities outside the blockchain.** Aramid prevents a compromised peer belonging to a shipping company from leaking the bid values to an outside entity with *network policies and policy checks at the trusted proxy.* Aramid's network rules restrict a compromised peer to interact with only the trusted proxy pod and smart contract pods with the network rules. It cannot send any data outside the network itself.

② **Sending unencrypted data.** Aramid uses TLS by default for all network connections, and this is enforced by the underlying service mesh. This attack is not possible on Aramid—all data leaving a pod is always encrypted.

③ **Intra-container data leaks.** When a container is misconfigured, or vulnerable Fabric components in it get exploited, the corresponding process can leak confidential information accessible to the container. Aramid follows standard security best-practices for Docker and Kubernetes and runs containers unprivileged, restricting file system access to read-only, and limiting interactions with the host.

④ **Inter-channel data leaks.** When a compromised Fabric peer is part of multiple channels, it can leak information from one channel to another. Aramid runs a separate peer instance per channel, each running on a different pod, and restricted in its communication using the network rules and pod security policies, thereby preventing the attack by construction.

⑤ **Leaking channel membership information.** In Aramid, a peer (or orderer) is part of only one channel and does not know about the existence of any other channels on the network. The *trusted proxy* handles new channel membership for a peer. While a peer has access to membership information for the channel it serves, it has no means of leaking this information to external entities.

⑥ **Leaking smart contract business logic.** In Aramid, smart contracts are deployed as external contracts, running in separate pods. A compromised smart contract cannot leak the business logic to anyone else but the peer instance on which it is deployed because it is restricted to communicate only to the peer instance. The compromised peer instance is restricted to communicate only to its trusted proxy, allowing Aramid to enforces policies to verify the message destination and content.

⑦ **Leak via non-channel messages.** In Fabric, a compromised peer can cleverly craft a non-channel gossip message with channel data and send it to peers on a different channel. In Aramid, the *trusted proxy* handles the non-channel gossip messages and responds on behalf of all the peer instances to prevent such attacks.

⑧ **Leaking private data stored in private data collection.** A malicious Fabric peer can send private data to peers that are not part of that collection. Aramid's proxy prevents this by checking the gossip message content and collection policy before forwarding the private data to other organizations.

4.6.2 Performance Evaluation

In this section, we compare the performance of Aramid with that of Fabric by varying several parameters: (1) the workload send rate; and (2) the number of channels.

The major difference between Fabric and Aramid is the addition of the trusted proxy in Aramid. Our experiments are designed to analyse the performance impact of adding this new network element on the overall performance of the blockchain. The throughput and latency numbers presented in this section are averaged over 10 runs. Each run completes 200,000 transactions.

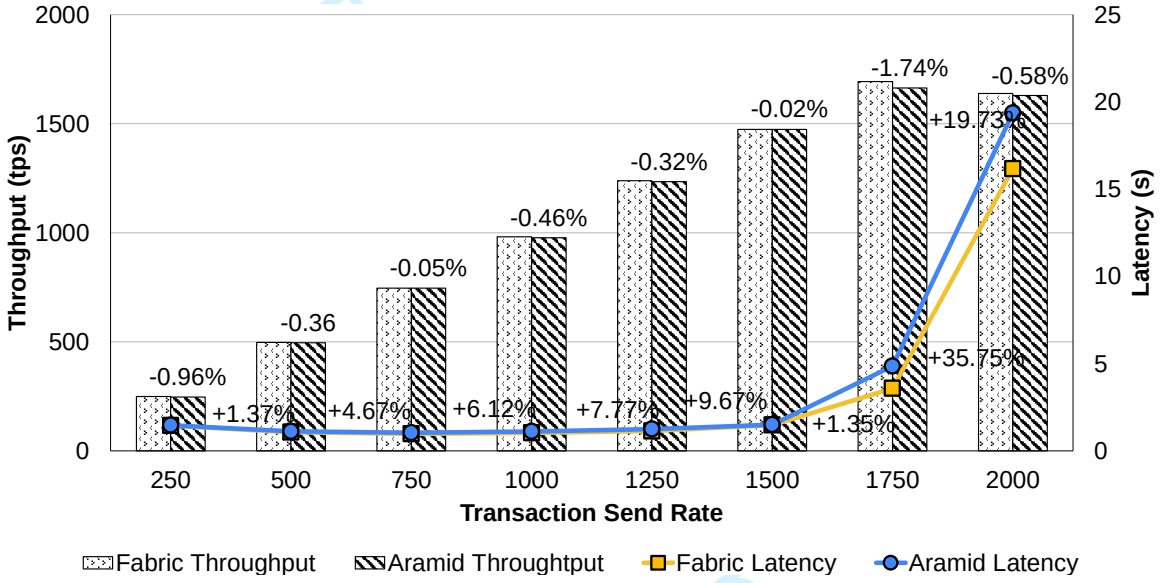

Figure 4.8: Measuring the impact of transaction send rate on throughput and latency in both Fabric and Aramid.

Impact of Transaction Send Rate Figure 4.8 plots the average throughput and latency achieved in Fabric and Aramid with different transaction send rates. Transactions are sent by Caliper workers. As our default endorsement policy is OR(Organization-1, Organization-2, Organization-3), *i.e.,* that the transaction must be endorsed by the membership service provider of any one of these organizations, Caliper is configured to send transaction proposals to only the peer of Organization-2 and get endorsements from it. Other parameters are configured as per Figure 4.6.

We observe that *the throughput profile of Fabric and Aramid closely match over varying transaction send rates.* With an increase in transaction-send rate the throughput increases linearly until it saturates at about 1670 transactions per second (tps) for both Fabric and Aramid.

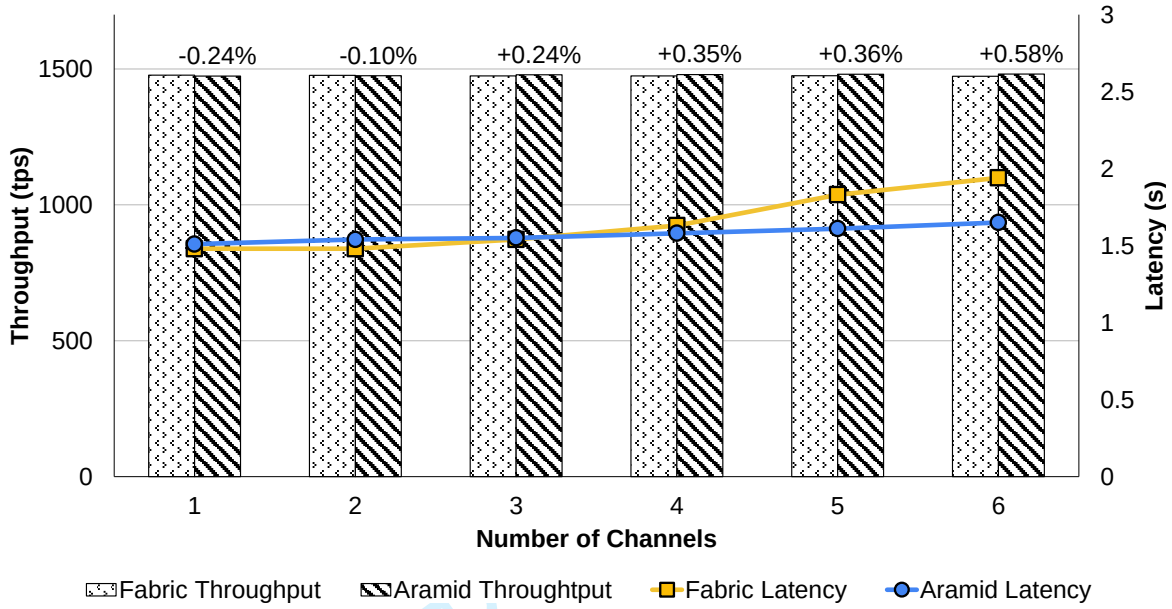

Figure 4.9: Measuring the impact of the number of channels on throughput and latency in Fabric and Aramid.

For lower send rates (250 tps and below) the latency is little higher than that of higher send rates. This can be attributed to cases where the send rate is below block size which is set to 500 transactions. This increases latency because the system waits for the block to be filled or until the block-creation timeout. Overall, however, our conclusion from this experiment is that the performance of Aramid compares favourably with that of Fabric. On average, Aramid throughput is 0.56% less than Fabric.

Impact of Multiple Channels Figure 4.9 plots impact of increasing the number of channels on the average throughput and latency in Fabric and Aramid. We create six channels between Organization-1 and Organization-2. We join the endorsing peer of Organization-2 on all channels. Caliper sends transactions simultaneously on multiple channels with a transaction send rate of 1500 tps evenly split among the available channels.

We first observe that *both Fabric and Aramid maintain nearly constant throughput and latency as the number of channels is increased.* As Figure 4.9 shows, the throughput is at around 1475tps and latency less than 2s. We note that *the throughput and latency of Aramid is slightly better than Fabric when number of channels increases.*

To study this trend in more detail, we conducted another experiment with six channels set and varied the transaction send rate. As Figure 4.10 shows, Fabric's throughput saturates at around 1950 tps, while Aramid's throughput saturates at 2100 tps. After the saturation point, the difference in the latencies increases. Aramid outperforms Fabric because it has a separate

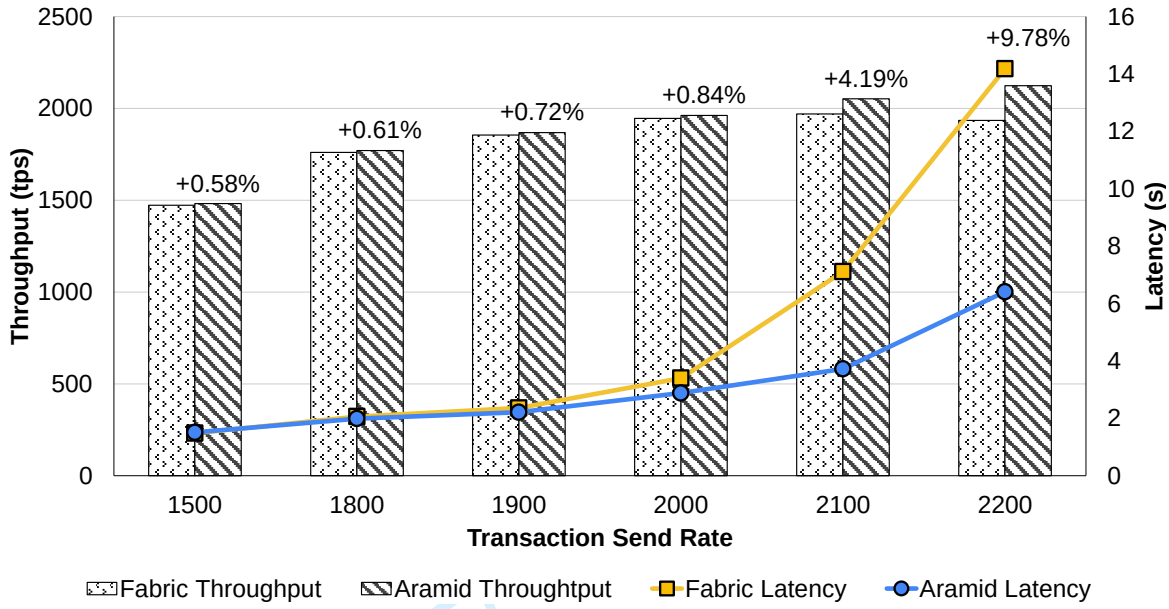

Figure 4.10: Fabric and Aramid with six channels.

peer instance and separate smart contract instance for each channel. Our conclusion from this experiment is that Aramid performs better when number of channels increases.

4.7 Summary

Aramid improves upon the channel-based data protection mechanisms of Fabric. Aramid's privilege-separated design allows us to relax the assumptions that Fabric makes about all participants being trusted, and provides channel-based data protection even in the presence of compromised peers, orderers and smart contracts. Aramid is transparent to end-user applications, thereby allowing interoperability with legacy Fabric applications, and provides transaction latency and throughput rates comparable to that of Fabric.

Chapter 5

Evaluation of Methods for Privacy Preserving Video Analytics

5.1 Introduction

While the PERQS in Chapter 3 facilitates distributed querying, it lacks a mechanism for privacy-preserving joint analysis. In this chapter, we address this challenge by evaluating two prominent paradigms for secure collaborative computation: Secure Multiparty Computation (MPC) [42, 172] and hardware-based Trusted Execution Environments (TEE) [110, 143]. We systematically compare these approaches in the context of realistic video analytics workloads to determine their feasibility, performance limits, and suitability for distributed surveillance environments.

MPC is a cryptographic approach that enables multiple parties to collaboratively compute a function over their inputs while keeping those inputs private. This method is entirely software-driven and does not require specialized hardware, making it versatile and applicable across diverse edge devices. Several previous works have used MPC for privacy-preserving data analytics, demonstrating its ability to securely process sensitive data without exposing raw inputs. On the other hand, TEE relies on secure hardware components to create isolated environments where computations can be performed securely, even if the host system is compromised. TEE has challenges, including hardware dependency, potential vulnerabilities due to side-channel attacks, and scalability limitations in highly distributed edge systems.

Both approaches are viable solutions for enabling secure and private computation in distributed environments. Each method has distinct strengths and limitations regarding performance, scalability, deployment complexity, and security guarantees. For example, MPC excels in settings with high distrust among parties but may struggle in low-bandwidth environments.

Conversely, TEE provides high computational efficiency but requires specialized hardware and may face adoption challenges due to hardware costs or vendor lock-in. In addition, MPC protocols offer a variety of design choices, including in-house vs. outsourced computation models, arithmetic vs. boolean, optimizations for 2 or 3-party settings, and different adversary models(active/passive). With recent advances and optimizations, their relative advantages, disadvantages, and suitability for privacy-preserving edge data analytics remain unclear. In this chapter we evaluate the relative merits of these two methods by analyzing their practical implementations and real-world performance of joint video analytics in edge computing environments. We will systematically assess the pros and cons of MPC and TEE across various metrics. The criteria on which we wish to evaluate the methods are:

- What are the computational and communication overhead? Performance evaluations to find which method is suitable for each application.
- What are the security implications of these methods? Strengths and vulnerabilities in protecting private data.
- What are the additional requirements to implement these solutions? Additional hardware or software required.
- How much effort is required to implement the solution? Ease of integration and deployment in edge devices.

By providing in-depth comparison and evaluation, we aim to clarify the trade-offs between MPC and TEE for privacy-preserving edge analytics and offer insights into which approach is better suited for different applications or constraints. This study will contribute to the advancement of secure edge analytics and help guide future developments in privacy-preserving technologies.

5.2 Secure Multiparty Computation (MPC)

Secure Multiparty Computation (MPC) [42, 172, 80, 57] is a cryptographic framework that enables a group of mutually distrusting parties to jointly compute a function over their private inputs without revealing those inputs to one another. This paradigm allows collaborative analytics and decision-making while preserving privacy—a property increasingly crucial in today's data-driven world, where data owners are often reluctant to share raw data due to privacy, regulatory, or competitive concerns. Unlike other privacy-preserving approaches, MPC operates without relying on a central trusted authority. Instead, the computation is distributed among the participating parties, and the protocols are designed to be resilient even in the presence of malicious actors who may attempt to disrupt the process or extract private information.

The idea of MPC was first introduced by Andrew Yao [42, 172] through the concept of the "millionaires' problem", which asked whether two millionaires could determine who is richer without disclosing their actual wealth. This pioneering problem set the stage for Yao's Garbled Circuits protocol, marking the origin of modern secure computation.

5.2.1 Cryptographic Foundations

MPC relies on a combination of cryptographic primitives to ensure privacy and correctness.

5.2.1.1 Secret Sharing

A foundational building block of many MPC protocols is secret sharing, where a value x is split into shares $x_1, x_2, ..., x_n$ distributed among parties such that: Any subset of shares below a threshold reveals nothing about x and the full set (or a qualified subset) can reconstruct x.

The most widely used scheme is Shamir's Secret Sharing (SSS)[42, 39], based on polynomial interpolation over finite fields. Additive secret sharing, a simpler variant, is also widely used in practical systems for efficiency.

5.2.1.2 Oblivious Transfer (OT)

OT allows a sender with two messages (m_0, m_1) to transfer one message m_b to a receiver without learning b, while the receiver learns nothing about m_{1-b}. OT is essential in Yao's Garbled Circuits and many other two-party computation protocols [172].

5.2.1.3 Garbled Circuits

In Yao's Garbled Circuits protocol, one party (the garbler) encrypts a boolean circuit representing the target function, while the other (the evaluator) computes the output without learning intermediate values. This method provides strong security guarantees but can be computation-heavy for large circuits. [172, 57]

5.2.1.4 Homomorphic Encryption

Homomorphic encryption allows computation directly on encrypted data. MPC protocols sometimes use partially or fully homomorphic encryption (FHE) schemes to perform additions and multiplications over ciphertexts, thereby avoiding decryption until the final step. [54, 154]

5.2.1.5 Zero-Knowledge Proofs (ZKPs)

Used in malicious-secure MPC to ensure that parties follow the protocol correctly without revealing their secrets. ZKPs can prove, for example, that a party's share is consistent with the expected computation [125, 47]

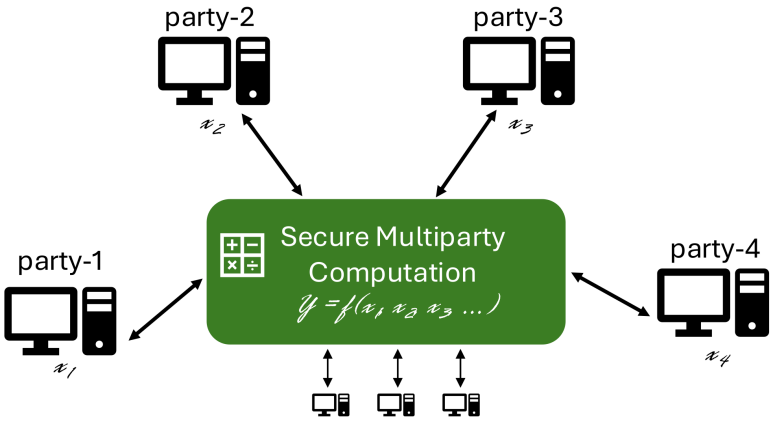

Figure 5.1: MPC protocol involves n parties, each holding private input x_i and computing a joint function $y = f(x_1, x_2, ..., x_n)$.

5.2.2 Fundamental Concepts

At its core, MPC involves n parties, each holding private input x_i and a joint function $f(x_1, x_2, ..., x_n)$ that they wish to compute [80] as shown in figure 5.1. An MPC protocol that ensures:

- Correctness: The output is the correct value of f.
- Privacy: No party learns anything more about others' inputs than what can be inferred from the output.
- Independence of Inputs: Inputs are chosen independently of protocol messages.
- Fairness (optional): Either all parties receive the output or none do.
- Robustness: The protocol tolerates some degree of malicious or faulty behavior.

MPC Protocols vary based on these fundamentals and security requirements, defending against semi-honest and malicious adversaries. Private inputs can be securely shared and computed using secret sharing and garbled circuits. Computation can be performed in-house or outsourced, with optimizations tailored for 2-party, or 3-party, or multi-party setups. While many variations have been explored and improved over the years. Some of the key ideas used in the design of a MPC protocols are:

① **Adversary Type (Active/Passive):** The Semi-honest(Passive) [58, 80] setting assumes that all participants follow the protocol but may attempt to infer private information from the data they receive. Provides weaker security guarantees but, in general is more efficient in terms of computation and communication overhead. In contrast, malicious(Active) [99, 80]

setting assumes that participants may deviate from the protocol, alter computations, or attempt to manipulate results to gain unauthorized information. It is more secure but has higher computational and communication overhead because of the additional security layers to protect against the malicious adversary.

② **Computation Domain (Arithmetic/Boolean):** Arithmetic MPC protocols (operating modulo a prime or power of two) [98] is more efficient for mathematical computations and real-world numerical applications. Boolean/Binary MPC protocols are better suited for logic-based operations but incurs higher communication overhead. Often implemented using garbled circuits and Oblivious Transfer (OT) [58].

③ **Computation Choice (Secret Sharing/Garbled Circuits):** In Secret Sharing [40] techniques, inputs are divided into "shares" distributed among the participants. Each share is meaningless but can reconstruct the input when combined with other shares. This ensures that no single party has access to the entire input. While Garbled Circuits [24, 172] represent a function as a circuit with encrypted values assigned to inputs and intermediate computations, ensuring that no private data is exposed during the computation process.

④ **Number of Parties (2-party/3-party/multi-party) :** Generally, 2-party setting [58] requires stronger cryptographic techniques since both parties must ensure privacy without a third-party mediator. Contrary 3-party setup [95] can leverage an honest majority for efficiency and security improvements. Typically, MPC protocols are inefficient as number of parties increases [57, 58].

⑤ **In-house/Outsourced:** In in-house MPC [95] all participating parties execute the MPC protocol on their infrastructure without relying on external computing resources. This is limited by the available hardware and network capacity of participating entities. In outsourced MPC model [53] major part of the computation is securely offloaded to external cloud servers. It is best suited for scenarios where participants have limited computational power, such as edge devices.

MPC enables privacy-preserving collaboration in distributed and edge computing environments by allowing multiple parties to jointly compute functions without revealing their private data [95, 53, 98]. Despite its strong privacy guarantees, MPC protocols often incur high communication and computation overhead due to frequent message exchanges and complex cryptographic operations, which can lead to increased latency in large-scale deployments. Ongoing research focuses on addressing these challenges through protocol optimizations, hybrid cryptographic–hardware approaches, and efficient system implementations, making MPC progressively more practical for real-world applications.

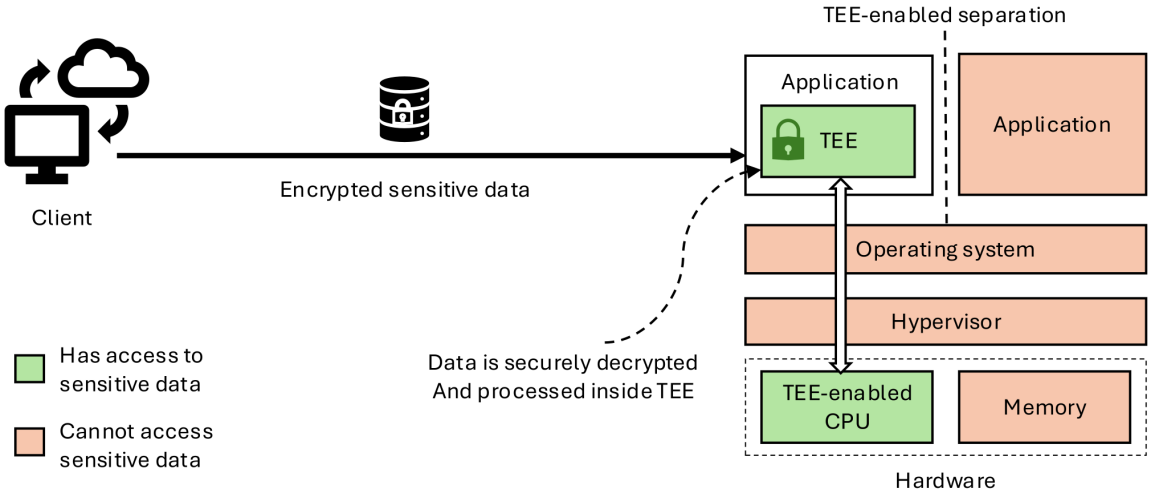

Figure 5.2: Client sends encrypted sensitive data to the trusted TEE application. Other applications, operating systems and even hypervisor cannot access the sensitive data.

5.3 Trusted Execution Environment (TEE)

A Trusted Execution Environment (TEE) [110, 143] is a secure area within a main processor that ensures the confidentiality and integrity of code and data loaded into it. It operates alongside the normal operating system but remains isolated from it, thereby protecting sensitive computations even if the rest of the system is compromised. TEEs have emerged as a key component in enabling hardware-assisted security, particularly for cloud computing, edge computing, and privacy-preserving data analytics.

The concept of TEE stems from the need to provide a trusted subsystem in an otherwise untrusted environment, where traditional software-based protections (like encryption or sandboxing) may not suffice. By isolating sensitive workloads in hardware-protected regions, TEEs enable secure computation, attestation, and data sealing — forming the foundation for trusted services such as confidential computing. Figure 5.2 demonstrate the use TEE for confidential computation. Some of the commonly available TEE implementations are as follows:

- Intel Software Guard Extensions (SGX) [38]: A hardware feature that provides secure enclaves for applications, enabling sensitive computations in an isolated memory region inaccessible to other processes or the OS.
- AMD Secure Encrypted Virtualization (SEV) [135]: A virtualization-based TEE solution that encrypts virtual machine memory to protect data from unauthorized access by hypervisors or other VMs.
- AWS Nitro [13, 21]: A virtualization-based TEE built on top of the AWS Nitro System.

Nitro enclaves run isolated from the host EC2 instance—no external networking, storage, or interactive access (e.g., SSH) is allowed.

• ARM TrustZone [18]: A system-wide approach that creates secure and normal worlds within a processor, allowing sensitive operations to run in an isolated environment.

Intel SGX creates secure enclaves for isolating small, sensitive workloads with strong security guarantees but limited memory and no system call support, making development complex. AWS Nitro Enclaves, in contrast, isolate larger workloads on EC2 instances, support integrated attestation and secure communication, and are easier to use in cloud environments.

5.3.1 Fundamental Concepts and Properties

A TEE provides a secure execution context within a processor, characterized by the following properties:

- **Isolation:** The TEE ensures strong hardware-level separation between the secure and normal worlds. Applications running inside the TEE cannot be accessed or modified by the host operating system, hypervisor, or other applications.
- **Integrity:** Code and data within the TEE cannot be tampered with by any entity outside the trusted boundary.
- **Confidentiality:** Data processed or stored inside the TEE is inaccessible to unauthorized software, even with root or kernel privileges.
- **Remote Attestation:** A cryptographic mechanism that allows a remote verifier to confirm the authenticity of the TEE and ensure it is running unmodified, trusted code.
- **Secure Storage (Sealing):** Data generated or used inside the TEE can be encrypted and stored securely for later use, bound to that specific TEE instance or device.
- **Controlled Invocation:** Only authorized entities can initiate execution inside the TEE, preventing unauthorized access to trusted resources.

A TEE operates as a co-processor abstraction within the main CPU. Architecturally, modern processors implement TEEs using a split-world model:

- **Normal World (Rich Execution Environment, REE):** Runs general-purpose applications and the main operating system. It is feature-rich but untrusted.

- **Secure World (TEE):** Runs a small, security-focused operating system (Trusted OS) and trusted applications. Access to memory and peripherals in this world is tightly controlled.

TEE bridge the gap between software-level and hardware-level security by creating isolated execution regions for sensitive computations. They provide confidentiality, integrity, and verifiability in untrusted environments such as the cloud and the edge. While limited by hardware constraints and side-channel vulnerabilities, continuous advances in TEE architectures and software ecosystems are making them central to the realization of confidential and privacy-preserving computing.

5.4 Overview

Sharing data between different parties like government agencies and private organizations raises serious privacy concerns. Video files are large, making secure sharing and joint processing more demanding than other types of data. On top of that, the complex machine learning models needed for video analysis make it harder to build systems that are both private and scalable. While prior work such as PERQS [74], Privid [31], RTFace [164], and VISOR [121] has explored secure video analysis, these systems often focus on local or isolated data processing. Our work builds upon PERQS [74], which uses local machine learning inference on CCTV streams to avoid raw video sharing but lacks support for joint and cross-organizational video analytics. To bridge this gap, we explore using MPC and TEE as foundational technologies for enabling privacy-preserving collaborative video analytics, offering strong security guarantees.

Our key contribution is identifying the most suitable implementation strategies for both approaches. In the case of MPC, a wide array of protocol variants exists. While prior research has explored general MPC performance characteristics, their suitability for collaborative video analytics tasks, particularly at the edge, remains underexplored. This work systematically examines the MPC design space to identify the most practical and efficient protocols for such workloads. In contrast, TEE-based approaches follow a more uniform design paradigm, where secure enclaves-whether through AMD-SEV, Intel-SGX, AWS-Nitro, or ARM TrustZone-process sensitive data in a hardware-isolated environment, ensuring confidentiality and integrity. Although conceptually similar, these implementations differ in their architectural design, affecting performance, integration complexity, and resilience to side-channel threats. To enable a robust evaluation, we design five real-world case studies that capture diverse challenges in joint video analytics, allowing us to analyze both MPC and TEE solutions regarding performance, security guarantees, and implementation complexity.

Case Study		Description	Critical Computation	Implementations	Datasets
①	Vehicle Re-Identification	identifying the same vehicle in another video camera.	Cosine Similarity Vector	DyeNet [92], PROVID [101], Siam R-CNN [156]	DAVIS dataset [124]
②	Scene Similarity	Detect weather two views are the same scene, or same events.	Euclidean Distance Measure	Factor Analysis [168], PDH Similarity [63],	Video Dataset [168]
③	Joint Recognition	Recognise a car which is partially visible in two video streams.	Video Stitching and Recognition	Object Centered Stitching [62]	Stitching Dataset [179]
④	Scene 3D Reconstruction	Merge different views from multiple cameras into a 3D scene and run query on it.	Scene 3D Reconstruction and query	EVolT [162], MR Video Fusion [188], DyNeRF [91]	Plenoptic Datasets [91], 3D Shapenets [170]
⑤	Count the Vehicles	From the overlapping video feeds find the number of vehicles in a certain interval.	Multi-video Object Counting	Argus [174], Rt3c [166]	CityFlow V2 [49], Cro HD dataset [145]

Table 5.1: Details of the case studies include their critical computations, which are to be evaluated under privacy-preserving conditions, along with the available implementations and datasets. We use CityFlowV2 [49] for our analysis.

5.4.1 Case Studies

We selected five case studies to capture a diverse range of challenges in joint video analytics, each representing distinct computational requirements and privacy constraints. The primary goal is to enable collaborative analytics without exposing raw video feeds, which often contain sensitive information. Table 5.1 summarizes the critical computations, implementations, and available datasets. Critical computations are offloaded to either the MPC or TEE frameworks, while other application parts are executed locally on edge devices. To ensure a fair evaluation, we maintain consistent implementation across both approaches. More details on each case study are discussed below.

① Vehicle Re-identification (Re-ID) [101, 92, 156, 177]: Vehicle re-identification aims to recognize the same vehicle across multiple cameras in a network, playing a vital role in traffic management, law enforcement, and stolen vehicle recovery. Previous approaches have used various features, such as vehicle color, make, model, and license plate patterns, for matching. We use a feature extractor that utilizes deep learning models to generate feature vectors locally at the edge devices. These feature vectors are then compared against other video feeds to identify matches. We employ the Cosine Similarity function to measure the similarity between feature vectors. Since vehicle features contain sensitive information, the joint execution of cosine

similarity must be conducted securely using MPC or TEE to preserve privacy.

② Scene Similarity Detection [187]: Scene similarity detection is crucial in video analytics to determine whether different cameras capture the same or similar environments. Since we do not trust any participants unquestioningly, the scene similarity check on overlapping scenes can reach a collective root of trust. Prior works [168, 63] explored various feature extraction methods for scene comparison. In our approach, we use a video summarizer to extract visual features as a vector, and then we use Euclidean distance to measure the distance between two scenes. We use the distance measure to compare and find the most similar past scenario for critical event detection. Scene features contain sensitive information and cannot be shared directly, so distance computation must be performed in a privacy-preserving manner.

③ Joint Object Recognition [128]: This case study focuses on recognizing objects, such as cars, bicycles, or trucks, which are spread across multiple cameras. Collaborative object recognition can enhance accuracy by combining video feeds from different sources securely [62]. For example, two cameras might capture complementary views of an object, improving performance. The videos are combined, and then object recognition is on top of the combined view. Here, the video stitching and recognition parts are the critical components that must be executed securely to protect video privacy.

④ Scene 3D Reconstruction [134, 188, 91, 162, 178]: Scene reconstruction involves piecing together video data from multiple cameras to create a comprehensive 3D view of a specific area or event. For instance, in the case of a traffic accident, video data from different angles can be collaboratively analyzed to reconstruct the sequence of events. For the experiment, we used a static angle, rule-based 3D reconstruction, and then performed queries on this reconstructed 3D scene. Since the input videos are private, 3D reconstruction and querying must be done securely.

⑤ Counting the Total Number of Vehicles [174, 166]: Vehicle counting is crucial in traffic management, urban planning, and intelligent transportation systems. Traditional methods rely on individual cameras counting vehicles independently, which may lead to duplicate counts when the exact vehicle appears in multiple camera feeds. In this case study, we detect and count vehicles locally at each camera and securely remove duplicates by comparing vehicle features from different camera feeds. Feature extraction is performed on each detected vehicle to generate a feature vector representing its characteristics. To eliminate duplicate counts, we employ Cosine Similarity to compare feature vectors across multiple camera detections. We use MPC or TEE to ensure that feature comparisons and duplicate removals are performed without exposing sensitive data, enabling privacy-preserving collaborative vehicle counting.

5.4.2 Threat Model

• **MPC Threat Model:** For evaluating MPC defines the following threat models: **Semi-Honest Model:** Where participants adhere to the protocol specifications but may attempt to learn additional information from received messages or intermediate computations. The system assumes that these participants do not alter their behavior maliciously. **Malicious Model:** Where participants may deviate from the protocol, intentionally sending incorrect messages or manipulating computations to compromise the integrity or confidentiality of the data. In both models, there can be an honest or dishonest majority. In an honest majority setting, the majority of participants are assumed to behave honestly during any protocol execution. In the context of our work, remote servers, if involved, are treated as semi-honest, meaning they follow the protocol but may try to infer private inputs.

• **TEE Threat Model** To evaluate TEE, we assume that secure hardware provided by third-party servers (either owned by participants or external entities) is considered trustworthy and free of back-doors or vulnerabilities. Participants encrypt their private inputs before sending them to the TEE-protected enclaves for processing. Computations are securely performed within the enclave; only the final results are shared with the participants. Potential attacks, such as side-channel attacks on TEE, network eavesdropping, or breaches of secure enclaves, are treated as out of scope.

Both threat models assume that communication channels are secure and that cryptographic measures are in place to protect data transmission. Physical access to devices or servers hosting the computation is assumed to be restricted. Any compromise of the underlying hardware or cryptographic primitives is beyond the scope of this evaluation.

5.5 Experiment Setup

This section outlines the experimental setup and benchmark details for evaluating privacy preserving video analytics methods. We aim to identify the most suitable options for secure video analytics at edge devices by benchmarking video processing workloads. The MPC protocols were evaluated across different configurations.

5.5.1 Setup for MPC

MPC configurations are evaluated on several key aspects. Depending on the security model, MPC can operate under a semi-honest or malicious setting. The number of participants also influences configuration choices, ranging from 2-party MPC, where two entities interact, to multiparty setups, such as 3-party MPC, which leverage an honest majority for efficiency. The

Protocol	GC /SS	Adversary Type	Honest Majority	Domain	#	Details
yao [172, 176]	GC	Semi-honest	No	Binary	2	half-gate garbling
real-bmr [82]	GC	Malicious	No	Binary	2+	MASCOT protocol using BMR
semi-bmr [23]	GC	Semi-honest	No	Binary	2+	semi-honest version of real-bmr
rep-bmr [23, 82]	GC	Semi-honest	Yes	Binary	3	replicated sharing using BMR
semi2k [96]	SS	Semi-honest	No	Mod 2^k	2	OT-based Beaver triples
semi-bin [96]	SS	Semi-honest	No	Binary	2	bit-wise multiplication triples
semi [96]	SS	Semi-honest	No	Mod Prime	2	OT-based prime Beaver triples
mama [83]	SS	Malicious	No	Mod Prime	2	MASCOT with several MACs
mascot [83]	SS	Malicious	No	Mod Prime	2	OT correlation checks & MACs
spdz2k [43]	SS	Malicious	No	Mod 2^k	2	more efficient than MASCOT
ring [97]	SS	Semi-honest	Yes	Mod 2^k	3	replicated ring shares
rep-field [97]	SS	Semi-honest	Yes	Mod Prime	3	replicated field shares
atlas [59]	SS	Semi-honest	Yes	Mod Prime	3+	ATLAS sharing
shamir [39, 36]	SS	Semi-honest	Yes	Mod Prime	3+	Shamir secret sharing
rep4-ring [97]	SS	Semi-honest	Yes	Mod 2^k	4	replicated sharing

- Number of participants supported, SS - Secret Sharing, GC - Garbled Circuit

Table 5.2: MPC protocols evaluated in this project. We use the MP-SPDZ [81] implementation.

details of the protocols we considered in our evaluations are in Table 5.2. The configuration options we considered in our evaluations are as follows:

① **Adversary Type (Active/Passive):** MPC protocols are available against semi-honest adversaries or malicious adversaries. Similarly, protocols that assume an honest majority leverage optimizations to enhance performance. To evaluate these claims, we explored different protocols that provide security against semi-honest and malicious adversaries, considering scenarios where an honest majority is present or not.

② **Computation Domain (Arithmetic/Boolean):** Arithmetic-based MPC protocols are generally more efficient for integer addition and multiplication tasks. In contrast, Boolean circuits can be better when computations involve logical operations. Given that our workloads consist of numerous integer and matrix operations, we conducted evaluations using image processing macro benchmarks to compare the performance of arithmetic-based and Boolean-based protocols.

③ **Computation Choice (Secret Sharing/Garbled Circuit):** Secret-sharing based computations require multiple rounds of communication, with complexity increasing depending on the operation. In contrast, garbled circuits execute computations in a single round but incur significantly higher costs for arithmetic operations due to their reliance on binary representation. Our evaluations aim to determine whether garbled circuits or secret sharing is superior to our workloads.

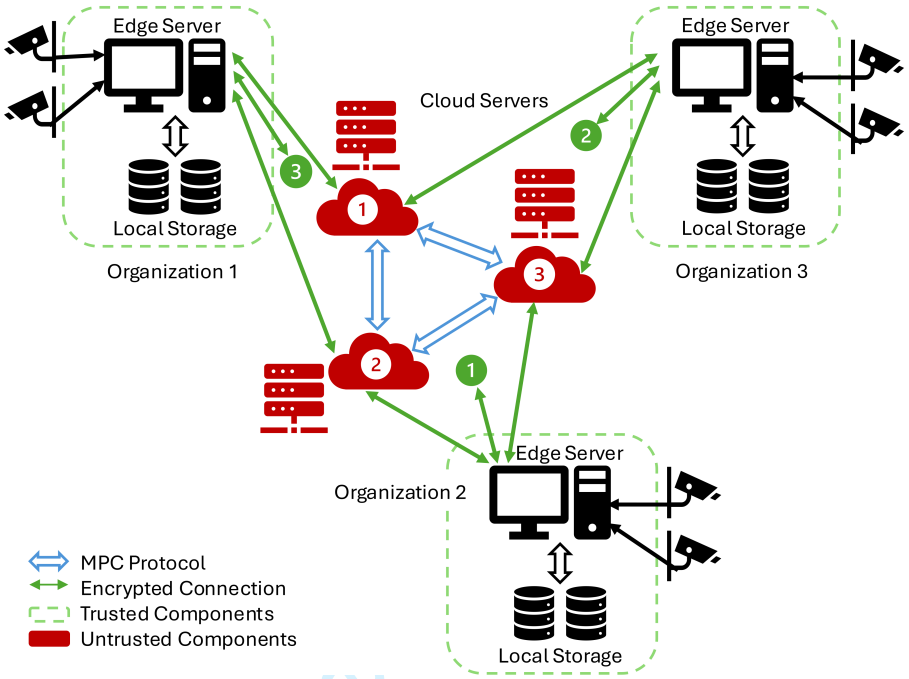

Figure 5.3: MPC outsourced model where secret shares of input is encrypted and send to the cloud servers. Then the servers will perform an MPC computation to evaluate the function on the secret input. This example is of a 3-party outsourced computation includes 3 independent servers.

④ **Number of Parties:** We evaluate the performance of the 2-party, 3-party, and 4-party protocols. Communication complexity generally increases as the number of participants increases. However, honest majority assumptions cannot be applied in a 2-party setting, while 3-party and higher setups can exploit these assumptions for performance optimizations. Since it remains unclear which n-party protocols will perform best for image processing workloads, we conduct extensive evaluations to determine the most suitable configuration.

⑤ **In-house/Outsourced model:** In an in-house MPC model shown in Figure 5.4, computations take place entirely within local edge devices. Alternatively, an outsourced MPC model shown in Figure 5.3 delegates computations to external servers. Our evaluation considers the trade-offs between these two approaches in the context of privacy-preserving video analytics.

5.5.2 Setup for TEE

We consider a cloud-based TEE architecture, as illustrated in Figure 5.5. In this setup, edge participants offload their computations to a secure enclave hosted on a remote server. Before initiating any computation, participating organizations verify the authenticity and integrity of the TEE hardware through remote attestation [38]. Once verified, they establish a secure,

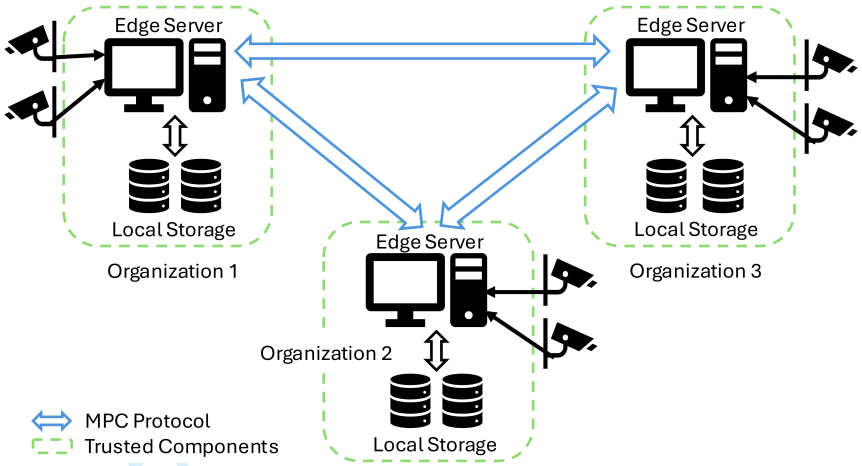

Figure 5.4: An example of 3-party setup for MPC in-house model where participants execute protocol among themselves. Local computation and intermediate communications are required depending on the protocol used.

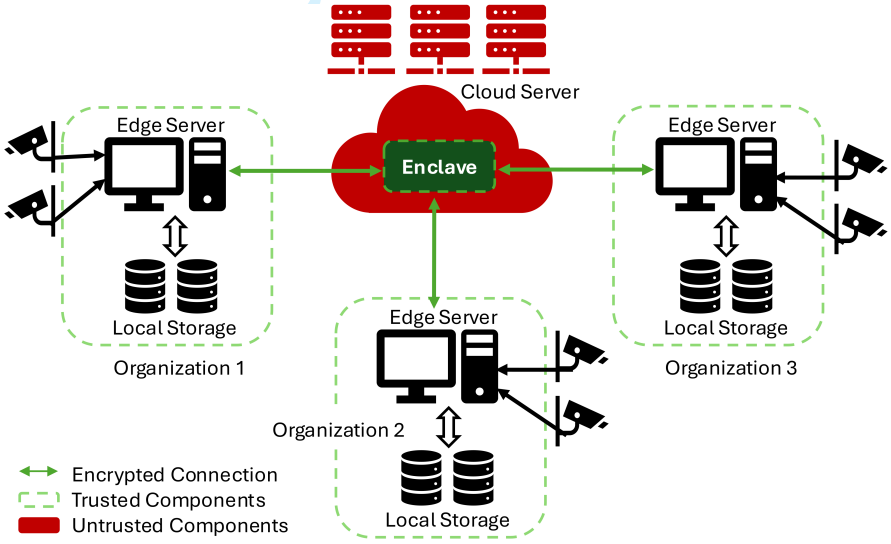

Figure 5.5: Three organizations perform collaborative data analytics using a trusted execution environment (TEE) available in a cloud server.

end-to-end encrypted connection directly to the enclave, ensuring that private data inputs are transmitted securely. This design prevents a compromised host operating system from accessing or extracting sensitive data. After collecting all video or feature inputs from the involved parties, the enclave executes the joint video analytics computation. Finally, the results are securely transmitted to the edge participants while maintaining data confidentiality.

Benchmarks: To evaluate the most suitable MPC protocol for secure video analytics, we use five image processing macro benchmarks [132]. These benchmarks-thresholding, histogram,

sobel edge detection, convolution, and thinning-represent fundamental operations commonly used in image and video processing. Each benchmark captures a distinct aspect of image analysis, such as feature extraction, edge detection, and transformation, making them well suited to assess the computational and communication overhead of different MPC protocols. We aim to identify the most reliable and efficient MPC protocol for edge devices by analyzing their performance.

Then for the remaining part of the TEE vs. MPC experiments, we focus on privacy-sensitive video analytics case studies, as explained in Table 5.1. We utilize the PERQS [74] framework to evaluate case studies, extending its query language to support joint video analytics. When a joint analysis query is executed, our MPC or TEE implementation is used to process the query securely. To compare the relative performance of both approaches across all five case studies, we measure execution time and global communication overhead for key computations involving private data. Beyond performance metrics, we also evaluate the security guarantees that each method provides and their practical implications. Additionally, we analyze the implementation complexity required to integrate MPC and TEE protocols into these applications securely.

5.6 Evaluation

This section outlines the implementation setup and the experimental evaluation. We begin by benchmarking various MPC protocols with image processing workloads. Followed by evaluating five case studies.

5.6.1 Implementation Details

We used the MP-SPDZ [81] framework for the implementation of MPC protocols, and Gramine-SGX [151, 152] and Linux-SGX [67] for Intel-SGX based TEE. All experiments were carried out on an Intel(R) Core(TM) i7-7700 CPU (3.60GHz) with four cores and two threads per core (8 hyper-threads), an 8192KB cache, and 32GB of RAM. Fedora 40 (Linux 6.13) was used for the MPC experiments, while Ubuntu 20.04 LTS (Linux 5.8) was used for the Gramine-SGX/Linux-SGX implementation. We also have an ARM-based AWS-Nitro-enabled c6g.large Linux instance with AWS-Amazon 2023 (Linux 6.1) for Nitro-based TEE evaluation.

For the evaluation of MPC protocols, we used five image processing macro operations [132]: thresholding, histogram, Sobel edge detection, convolution, and thinning, along with a simple matrix multiplication workload. These benchmarks represent core computational tasks commonly used in image and video processing. For the experimental validation of our five case studies, we used the AICITY21 benchmark dataset (CityFlowV2) [49, 35], which features real-world traffic surveillance data collected from 46 cameras. The dataset contains 880 anno-

Applications	Secret Sharing Based				Garbled Circuit		Malicious Adversary	
	semi2k	semi-bin	ring*	rep-field*	yao	rep-bmr*	mascot	spdz2k
Matrix Operations	**0.03 s**	0.18 s	0.03 s	**0.02 s**	0.11 s	1.78 s	1.64 s	1.49 s
	0.5 mb	4.3 mb	**0.02 mb**	0.26 mb	9.12 mb	520 mb	352.9 mb	277.4 mb
Thresholding	**1.36 s**	2.61 s	**0.08 s**	0.43 s	1.75 s	39.12 s	569.3 s	232.7 s
	202.9 mb	**46.5 mb**	**8.1 mb**	26.2 mb	83.9 mb	4.90 gb	104.9 gb	40.8 gb
Histogram	**1.93 s**	3.08 s	**0.07 s**	0.456 s	2.12 s	46.8 s	488.4 s	181.7 s
	278.8 mb	**103.5 mb**	**9.8 mb**	24.81 mb	268.3 mb	4.25 gb	94 gb	30.3 gb
Sobel	**27.2 s**	NA	**4.23 s**	4.36 s	NA	NA	568.3 s	450.8 s
	7.23 gb	NA	**28.8 mb**	57.67 mb	NA	NA	135 gb	118.1 gb
3x3 Convolution	**13.9 s**	NA	4.43 s	**4.12 s**	NA	NA	354.3 s	274.1 s
	3.56 gb	NA	**14.5 mb**	28.92 mb	NA	NA	66.5 gb	58.1 gb
Thinning	**4.96 s**	7.85 s	**0.11 s**	0.58 s	16.8 s	198.6 s	610.6 s	278.9 s
	1.25 gb	967.8 mb	**22.1 mb**	54.17 mb	2.34 gb	326.4 gb	124.3 gb	58.2 gb

* 3-party protocols

Table 5.3: MPC protocols performance varying sharing scheme and adversary type. We have measured execution time and global data sent for all protocols. Some results are unavailable because the MP-SPDZ [81] does not support it. The highlighted values are the best time and global data communication for 2-party and 3-party protocols.

tated vehicles in six different scenarios, with 215.03 minutes of video footage. We focused on eight junctions where multiple cameras provide overlapping recordings, enabling comprehensive multi-camera analysis.

5.6.2 Choosing the Best MPC Protocol

We utilized six workloads for our evaluation, matrix operations (including simple matrix additions and multiplications) and five fundamental image processing functions. To evaluate the image processing functions, we used a 256×256 monochrome image divided into four equal 128×128 pieces, each assigned to a different participant. For the 2-party setting, each party received two pieces of the image, while in the 4-party setting, each participant held one piece of the image. Then, a joint image processing operation was performed across all pieces, ensuring collaborative analysis while maintaining privacy. The MPC protocols listed in Table 5.2 were evaluated for these computations, with detailed results provided in Table 5.3 and Table 5.4. We have measured the total execution time, including the online and offline phases and the total global data communication. The MPC evaluation concerning the different design configurations is as follows.

① **Adversary Type (Active/Passive):** Malicious-secure protocols require significantly more execution time and communication overhead. As the complexity of operations increases, the communication cost grows exponentially. For example, in Sobel edge detection, all three malicious-secure protocols required over 100 GB of data exchange, making them impractical

Applications	2-party		3-party			4-party		
	semi2k	semi	ring	atlas	shamir	rep4-ring	atlas	shamir
Matrix Operations	**0.03 s**	0.11 s	**0.03 s**	0.06 s	0.04 s	0.07 s	0.08 s	0.05 s
	0.5 mb	15.9 mb	**0.02 mb**	0.88 mb	0.56 mb	0.3 mb	1.95 mb	1.12 mb
Thresholding	1.36 s	2.46 s	**0.08 s**	6.03 s	3.10 s	0.15 s	6.65 s	3.65 s
	202.9 mb	258 mb	**8.08 mb**	105.1 mb	86.5 mb	20.3 mb	207.8 mb	170.5 mb
Histogram	1.93 s	2.17 s	**0.07 s**	4.81 s	3.32 s	0.18 s	5.64 s	4.03 s
	278.8 mb	194 mb	**9.8 mb**	89.7 mb	75 mb	23.7 mb	177 mb	147.4 mb
Sobel	27.2 s	94.4 s	**4.23 s**	14.9 s	7.19 s	6.52 s	24.5 s	9.72 s
	7.23 gb	21.5 gb	**28.8 mb**	76.6 mb	57.7 mb	57.7 mb	153.1 mb	115.3 mb
3x3 Convolution	13.9 s	53.5 s	**4.43 s**	13.81 s	6.62 s	6.18 s	23.1 s	8.84 s
	3.56 gb	10.6 gb	**14.5 mb**	38.4 mb	28.9 mb	28.9 mb	76.9 mb	57.8 mb
Thinning	4.96 s	16.1 s	**0.11 s**	6.14 s	3.56 s	0.34 s	7.75 s	4.15 s
	1.25 gb	3.39 gb	**22.1 mb**	141.3 mb	114.5 mb	47.2 mb	278.2 mb	224.3 mb

Table 5.4: MPC protocol performance when number of parties increases.

for complex video analytics workloads.

② **Computation Domain (Arithmetic/Boolean):** Arithmetic-based protocols consistently outperform Boolean-based protocols in image/video processing tasks, as they involve extensive matrix operations. Boolean protocols are more efficient for numerous comparisons (e.g., thresholding) operations. For thresholding, the *semi-bin* protocol exhibited a lower communication overhead than *semi2k*, but for all other workloads, *semi2k* outperformed *semi-bin*. Additionally, MP-SPDZ currently does not support direct matrix multiplication for binary-shared inputs, preventing the execution of two workloads-Sobel and Convolution.

③ **Computation Choice (Secret Sharing/Garbled Circuit):** Secret Sharing based protocols consistently achieve lower execution time and reduced communication overhead across all workloads. As the complexity of integer operations increases, secret sharing is significantly more efficient than garbled circuits, which struggle with arithmetic intensive computations.

④ **Number of Parties:** The 3-party replicated secret-sharing protocol (*ring*) achieved the fastest execution time and lowest communication overhead across all workloads. The 2-party setting lacks optimizations available in 3-party honest-majority settings, making it less efficient for our use case. As the number of parties increases, both execution time and global data communication also increase.

⑤ **In-house/Outsourced model:** In our protocol, edge devices participate in computations but have limited resources and infrastructure. An outsourced model, as illustrated in Figure 5.3, shifts the computation to cloud servers. Here, edge devices only share input secret shares, significantly reducing their communication and computation load. This is particularly advantageous for DVR recorders and smart cameras, which lack the processing power required

Case Study		TEE (Intel SGX)		TEE (AWS-Nitro)		MPC (MP-SPDZ)	
①	Vehicle Re-Identification	0.002 s	98.2 kb	0.001 s	98.2 kb	0.006 s	218.2 kb
②	Scene Similarity	0.003 s	123.4 kb	0.0012 s	123.4 kb	0.015 s	0.36 mb
③	Joint Recognition	NA	NA	4.1 s	37.3 mb	347.53 s	5.08 gb
④	Scene 3D Reconstruction	NA	NA	7.3 s	34.3 mb	621.4 s	8.56 gb
⑤	Count the Vehicles	0.025 s	1.3 mb	0.017 s	1.3 mb	0.033 s	2.1 mb

Table 5.5: Performance of joint video analytics tasks of MPC and TEE. We used videos from CityFlowV2 [35] for all five workloads. Because of the lack of support and inherent memory limitations we are unable to run ml models using Intel SGX.

for joint video analytics using MPC.

Based on our evaluations, the Secret-Sharing-based 3-party replicated protocol (***ring***) is the most suitable choice for privacy-preserving video analytics workloads. We will use this protocol for our comparative analysis against TEE-based approaches.

5.6.3 MPC vs TEE Performance

In Case Study 1 (Vehicle Re-identification), vehicle features are extracted locally at the edge devices, followed by a privacy preserving computation of cosine similarity to identify matching vehicles across multiple cameras. For Case Study 2 (Scene Similarity), scene-level video features are extracted at the edge, and the Euclidean distance between these features is computed securely. Case Study 3 (Collaborative Object Recognition) combines two video streams using a predefined, rule-based stitching approach to generate a composite video, which is then processed using a vehicle recognition model. In Case Study 4 (3D Scene Reconstruction and Query), an ML model takes two input video streams to reconstruct a 3D scene representation. A subsequent query is then performed to detect specific objects within the reconstructed 3D environment. In Case Study 5 (Vehicle count), vehicles are first detected locally, and their features are extracted; the secure computation is then used to identify and eliminate duplicate detections across multiple camera feeds, ensuring an accurate count. For this evaluation, we used videos from the CityFlowV2 dataset [35], trimming them to similar lengths to ensure uniformity. We then averaged the execution time and communication cost across different sets of inputs.

The experimental results, as shown in Table 5.5, highlight the contrasting performance characteristics of MPC, and TEE-based implementations. For relatively lightweight workloads-such as vehicle re-identification (Case Study 1), scene similarity detection (Case Study 2), and total

vehicle count (Case Study 5)-MPC protocols performed reasonably well. On average, they were 3 to 6 times slower than their TEE counterparts and incurred about twice the communication overhead. We extract features locally for these three case studies, so the computational cost of secure execution is relatively contained, making MPC a viable option despite its performance penalties.

In contrast, Case Studies 3 and 4, which require collaborative execution of deep learning models for object recognition and video scene stitching, posed significant challenges for MPC frameworks. These workloads demand intensive matrix operations, activation functions, and large intermediate state sharing, which are expensive in MPC due to multiple communication rounds and complex arithmetic over shared values. As a result, MPC implementations were observed to be 70 to 90 times slower than TEE-based solutions. Moreover, the communication overhead ballooned into gigabytes, making the approach infeasible for real-time or bandwidth-constrained environments. However, TEE-based solutions efficiently handled these complex ML workloads, with only a few megabytes of private video input securely transferred to the enclave. Once inside, the entire model execution occurred at near-native speeds, providing a significant performance advantage. These results underscore the practical limitations of MPC in high-computation scenarios and demonstrate the strengths of TEE in handling complex, resource-intensive tasks.

5.6.4 Security Evaluation

MPC provides a robust and flexible security model that operates entirely in software and does not rely on trusted hardware components. This allows it to be deployed across a wide range of environments without dependence on specific hardware vendors or platforms. In addition, MPC protocols can be tailored to different threat models. The semi-honest model assumes that participants follow the protocol, and more secure configurations, such as protecting against malicious adversaries, can be adopted when stronger guarantees are required. These configurations introduce additional cryptographic checks and redundancy mechanisms, which increase computational and communication overhead but provide significantly enhanced security assurances.

In contrast, TEE such as Intel SGX or AWS Nitro Enclaves rely on secure hardware components to protect computations and data. TEE creates isolated enclaves where sensitive data can be processed securely, offering strong protection against software-level attacks and even against a compromised host operating system. However, their security is tied to the correctness and integrity of the hardware and its firmware, which introduces a dependency on hardware vendors. Establishing a root of trust with the manufacturer is necessary, and any vulnerabilities in the TEE implementation, such as side-channel attacks, speculative execution flaws (e.g.,

Spectre/Meltdown), or leakage through power analysis, can undermine the security guarantees. These limitations have led to concerns about TEE resilience.

5.6.5 Implementation Challenges

The MP-SPDZ framework offers a comprehensive and well-documented suite of tools to implement various MPC protocols. Its modular architecture and support for both arithmetic and Boolean computation domains make it suitable for rapid prototyping and evaluation. However, implementing machine learning (ML) workloads within this framework presents additional challenges. Specifically, developers must explicitly define the model architecture, layers, and activation functions compatible with the MPC back-end, which can be time-consuming and error-prone.

On the TEE side, Intel SGX poses several practical limitations. Most notably, SGX does not support system calls within the enclave environment, severely restricting the use of standard libraries and pre-existing machine learning frameworks such as PyTorch or TensorFlow. This limitation complicates the implementation of complex workloads that rely on dynamic memory management or file I/O. Furthermore, Intel has officially discontinued support for SGX on consumer platforms, reducing its viability for long-term and large-scale deployments. SGX enclaves also suffer from a memory limit (128MB of EPC memory), significantly hindering the execution of modern ML models that typically require larger memory footprints. Due to these limitations, we could not implement Python-based ML inference inside Intel SGX for our evaluation.

In contrast, AWS Nitro Enclaves provide a more flexible and developer-friendly environment. Nitro uses a container-based approach, allowing users to deploy secure enclaves by packaging their applications into Docker images. Which simplifies the enclave deployment process, especially for complex applications that depend on a large software stack. Additionally, Nitro Enclaves benefits from the scalability of the underlying EC2 infrastructure, users can allocate more CPU or memory resources to the enclave by selecting appropriate instance types. This makes it feasible to run memory-intensive ML workloads securely without modifying the application logic extensively.

5.7 Summary

We comprehensively evaluated Secure Multiparty Computation (MPC) and Trusted Execution Environments (TEE) as privacy-preserving solutions for joint video analytics in this work. We evaluated both approaches using five real-world case studies from traffic surveillance scenarios in the CityFlowV2 dataset. The case studies encompass a range of tasks, from simple simi-

larity computations to complex multi-camera machine learning-based analytics, allowing us to evaluate the strengths and limitations of both security paradigms systematically.

Our experimental findings show that TEE consistently outperforms MPC implementations for workloads involving machine learning inference. This performance gap is particularly notable in case studies involving deep learning models. TEE-based implementations achieved up to 70-90× lower latency and required only a fraction of the communication overhead compared to MPC. TEE benefits from a centralized and hardware-isolated execution model, which enables efficient data handling and reduced interaction during computation. However, TEE is not without limitations. They require trusted hardware support and rely on vendor-specific root-of-trust mechanisms, which may not be feasible in all deployment environments. Additionally, their susceptibility to side-channel attacks and ongoing hardware vulnerabilities raises concerns about their suitability for highly sensitive applications. In contrast, MPC offers a software-only solution that can be deployed in environments where trusted hardware is unavailable or undesirable. Among the MPC protocols evaluated, secret-sharing-based 3-party protocols - particularly those exploiting honest majority assumptions - demonstrated the best performance in terms of both execution time and communication cost. While MPC remains less efficient than TEE for ML-based video analytics, it provides stronger fault isolation and more transparent trust assumptions.

In conclusion, while TEE is currently the preferred choice for compute-intensive video analytics tasks due to its performance benefits, MPC offers a viable and secure alternative for resource-constrained or hardware-agnostic settings. Selecting between the two approaches depends on the application environment's specific workload, deployment constraints, and trust model.

Chapter 6

Conclusions and Future Directions

6.1 Conclusion

This thesis addresses a fundamental systems challenge: how to enable collaborative video analytics across mutually untrusted stakeholders without sacrificing privacy, integrity, or scalability. Rather than treating trust, infrastructure security, and joint computation as isolated problems, this work approaches them as interdependent layers of a single architecture.

At the application layer, PERQS establishes a trustless querying model in which distributed camera networks can collectively answer surveillance queries without centralized control. By embedding consensus directly into the analytics pipeline and anchoring commitments on a blockchain, the framework ensures that event detection is verifiable, tamper-evident, and resilient to faulty or malicious participants. This transforms video surveillance from a siloed, organization-centric model into a coordinated, query-driven ecosystem.

However, consensus and blockchain commitments alone are insufficient if the underlying infrastructure can leak data across administrative boundaries. This motivates the second layer of the thesis: strengthening the confidentiality guarantees of the permissioned blockchain itself. The privilege-separation architecture ensures that organizational boundaries enforced at the logical level are also enforced at the execution level. By reducing the trusted computing base and isolating channel components, the system aligns infrastructure-level protections with the higher-level privacy goals of collaborative analytics, thereby closing a critical gap between theoretical data isolation and practical deployment security. At the same time, Aramid is designed to reduce the attack surface of permissioned blockchain deployments rather than eliminate all forms of data leakage. In particular, it does not prevent out-of-band disclosure by malicious business insiders who possess legitimate access to channel data. Instead, Aramid confines trust to a smaller infrastructure-level enforcement layer and minimizes opportunities

for unauthorized cross-channel exposure within the blockchain execution environment.

The third layer addresses a remaining limitation: how to perform joint analytics across organizations when raw video cannot be shared. Here, the comparative study of MPC and TEE extends the framework beyond query coordination to secure cross-party computation. The evaluation demonstrates that no single technique universally dominates. TEE enables near-native performance for compute-intensive workloads, making real-time collaborative machine learning feasible. MPC, while significantly slower for heavy workloads, offers stronger decentralization and eliminates reliance on hardware trust anchors. Together, these results define a clear decision boundary: hardware-assisted isolation is preferable when performance is paramount, whereas cryptographic collaboration is suitable when trust minimization is the primary objective. Viewed collectively, the three contributions form a coherent architecture:

- Consensus mechanisms ensure correctness and accountability of distributed analytics.
- Infrastructure-level privilege separation ensures confidentiality and fault containment.
- Secure computation techniques enable privacy-preserving joint processing across organizational boundaries.

Each component addresses a distinct layer of the trust stack- application-level validation, system-level isolation, and computation-level privacy. Their integration provides end-to-end guarantees that neither component could achieve independently.

This layered synthesis advances the design of collaborative surveillance systems from ad hoc data sharing toward principled, verifiable cooperation. By quantifying performance trade-offs and demonstrating bounded overheads, the thesis shows that strong security properties-trustlessness, tamper resistance, confidentiality, and privacy-preserving computation-can coexist with practical scalability.

Ultimately, this work lays the foundation for building multi-stakeholder video analytics platforms that are not only technically secure, but also structurally trustworthy. It demonstrates that accountable collaboration in surveillance systems is achievable when consensus, infrastructure hardening, and secure computation are designed as complementary components of a unified architecture rather than as isolated solutions. At the same time, the proposed mechanisms operate under practical constraints. In particular, time-travel consensus is not universally applicable to all event types and depends heavily on sufficient spatiotemporal continuity and camera density. Its effectiveness is therefore bounded by the availability of nearby camera coverage and the ability to correlate events across space and time.

6.2 Future Directions

This thesis opens up several promising avenues for future research and system enhancement in the areas of secure, distributed, and privacy-preserving video analytics. While PERQS, Aramid, and the comparative evaluation of MPC and TEE collectively provide a strong foundation for secure collaborative surveillance, several extensions can further improve scalability, adaptability, and security assurance.

A natural next step is to extend PERQS into a fully decentralized edge–cloud federation, where edge devices autonomously participate in secure video analytics without relying on centralized coordination. Integrating federated learning techniques with PERQS could enable model sharing and collaborative training across distributed cameras while maintaining strict data privacy. This direction would allow continuous learning from real-world video streams while minimizing communication overhead and improving adaptability to new environments or event types.

Another significant research opportunity lies in developing hybrid computation frameworks that combine the strengths of MPC and TEE. Such systems could dynamically select the most appropriate secure execution model depending on the workload characteristics, network conditions, and trust assumptions. For instance, lightweight pre-processing could be performed using MPC at the edge, followed by deep learning inference within TEE enclaves for efficiency. Designing adaptive protocols for secure task partitioning across MPC and TEE layers remains an open challenge that could yield substantial performance and security gains. On the analytics side, expanding PERQL, the query language of PERQS, to support temporal, semantic, relational join, and multi-modal queries presents a valuable extension. Such capabilities would allow richer event analysis—such as correlating vehicle trajectories across time, performing join-style queries across multiple camera streams and intersections, integrating textual sensor metadata, or fusing multiple data modalities like video, LiDAR, and audio. For example, future extensions could support queries that trace an object appearing at one junction and subsequently reappearing at another within a specified time window. Integrating state-of-the-art AI-driven video understanding models and optimizing them for secure execution within MPC or TEE contexts would further improve analytical accuracy while maintaining privacy guarantees. Finally, a long-term direction involves real-world deployment and user studies. Deploying PERQS in collaboration with city authorities or campus-scale networks would provide valuable insights into operational challenges, regulatory compliance, and user trust. Addressing these socio-technical factors-such as transparency, accountability, and fairness-will be crucial for the adoption of secure surveillance frameworks in smart cities.

Bibliography

[1] Couchdb name collisions due to similar named channels, February 2017. https://jira.hyperledger.org/browse/FAB-2487. 60

[2] Cwe: Cwe-502 vulnerability in deserialization of untrusted data in google guava, November 2018. https://jira.hyperledger.org/browse/FABJ-390. 60

[3] Fabric sdk node memory exposure vulnerability, July 2019. https://jira.hyperledger.org/browse/FABN-1290. 60

[4] API endpoints are vulnerable to SQL injection, September 2020. https://jira.hyperledger.org/browse/BE-833. 60

[5] ECDSA signature validation vulnerability by accepting wrong ASN.1 encoding in jsrsasign, June 2020. https://jira.hyperledger.org/browse/FABN-1585. 60

[6] 1211. An overview of video analytics in security, 2022. https://www.ifsecglobal.com/video-surveillance/overview-video-analytics-security/ (accessed on 23 October 2023). 1

[7] 1231. Surveillance camera statistics: which cities have the most cctv cameras?, July 2022. https://www.comparitech.com/vpn-privacy/the-worlds-most-surveilled-cities/ (accessed on 23 October 2023). xi, 2

[8] 1234. Global surveillance camera markets 2021-2025, August 2021. https://www.prnewswire.com/news-releases/global-surveillance-camera-markets-2021-2025---integration-of-ai-systems-adoption.html (accessed on 23 October 2023). 1

[9] 1241. Adoption of advanced video analytics is "rising rapidly", nw security uk study finds, January 2021. https://www.ifsecglobal.com/video-surveillance/

BIBLIOGRAPHY

adoption-of-advanced-video-analytics-is-rising-rapidly-nw-security-uk-study-finds (accessed on 23 October 2023). 1

[10] 1251. India cctv market - growth, trends, covid-19 impact, and forecasts (2022 - 2027), August 2021. https://www.mordorintelligence.com/industry-reports/india-cctv-market (accessed on 23 October 2023). 1

[11] Devdatta Akhawe, Prateek Saxena, and Dawn Song. Privilege separation in html5 applications. In *Presented as part of the 21st {USENIX} Security Symposium ({USENIX} Security 12)*, pages 429–444, 2012. 13

[12] Devdatta Akhawe, Frank Li, Warren He, Prateek Saxena, and Dawn Song. Data-confined html5 applications. In *European Symposium on Research in Computer Security*, pages 736–754. Springer, 2013. 13

[13] Amazon Web Services. Aws nitro enclaves. https://docs.aws.amazon.com/enclaves/latest/user/nitro-enclave.html, 2020. Accessed: 2025-04-11. 83

[14] Elli Androulaki, Artem Barger, Vita Bortnikov, Christian Cachin, Konstantinos Christidis, Angelo De Caro, David Enyeart, Christopher Ferris, Gennady Laventman, Yacov Manevich, Srinivasan Muralidharan, Chet Murthy, Binh Nguyen, Manish Sethi, Gari Singh, Keith Smith, Alessandro Sorniotti, Chrysoula Stathakopoulou, Marko Vukolić, Sharon Weed Cocco, and Jason Yellick. Hyperledger fabric: A distributed operating system for permissioned blockchains. In *Proceedings of the Thirteenth EuroSys Conference*, EuroSys '18, New York, NY, USA, 2018. Association for Computing Machinery. ISBN 9781450355841. doi: 10.1145/3190508.3190538. URL https://doi.org/10.1145/3190508.3190538. 53, 55

[15] Apache. Apache couchdb, 2021. URL https://couchdb.apache.org/. 70

[16] Guilherme H. Apostolo, Pablo Bauszat, Vinod Nigade, Henri E. Bal, and Lin Wang. Live video analytics as a service. In *Proceedings of the 2nd European Workshop on Machine Learning and Systems*, EuroMLSys '22, page 37–44, New York, NY, USA, 2022. Association for Computing Machinery. ISBN 9781450392549. doi: 10.1145/3517207.3526973. URL https://doi.org/10.1145/3517207.3526973. 5, 12

[17] Walid Aref, Moustafa Hammad, Ann Christine Catlin, Ihab Ilyas, Thanaa Ghanem, Ahmed Elmagarmid, and Mirette Marzouk. Video query processing in the vdbms testbed for video database research. In *Proceedings of the 1st ACM International Workshop on*

BIBLIOGRAPHY

Multimedia Databases, MMDB '03, page 25–32, New York, NY, USA, 2003. Association for Computing Machinery. ISBN 1581137265. doi: 10.1145/951676.951682. URL https://doi.org/10.1145/951676.951682. 5, 12

[18] Arm. Arm platform security architecture. *Arm Limited*, 2019. 84

[19] Mohamed Azab and Mohamed Eltoweissy. Migrate: Towards a lightweight moving-target defense against cloud side-channels. In *2016 IEEE Security and Privacy Workshops (SPW)*, pages 96–103. IEEE, 2016. 14

[20] Richard Barker. How many cctv cameras are in london?, Sept. 2022. https://clarionuk.com/resources/how-many-cctv-cameras-are-in-london/ (accessed on 23 October 2023). 1

[21] Jeff Barr. Introducing aws nitro enclaves. https://aws.amazon.com/blogs/aws/introducing-aws-nitro-enclaves/, 2020. Accessed: 2025-04-11. 83

[22] Favyen Bastani, Songtao He, Arjun Balasingam, Karthik Gopalakrishnan, Mohammad Alizadeh, Hari Balakrishnan, Michael Cafarella, Tim Kraska, and Sam Madden. Miris: Fast object track queries in video. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, SIGMOD '20, page 1907–1921, New York, NY, USA, 2020. Association for Computing Machinery. ISBN 9781450367356. doi: 10.1145/3318464.3389692. URL https://doi.org/10.1145/3318464.3389692. 12

[23] Donald Beaver, Silvio Micali, and Phillip Rogaway. The round complexity of secure protocols. In *Proceedings of the 22nd annual ACM symposium on Theory of computing*, pages 503–513, 1990. 89

[24] Mihir Bellare, Viet Tung Hoang, and Phillip Rogaway. Foundations of garbled circuits. In *Proceedings of the 2012 ACM conference on Computer and communications security*, pages 784–796, 2012. 82

[25] Eli Ben-Sasson, Alessandro Chiesa, Christina Garman, Matthew Green, Ian Miers, Eran Tromer, and Madars Virza. Zerocash: Decentralized anonymous payments from bitcoin. In *IEEE Symposium on Security and Privacy*, 2014. 15, 52

[26] F. Benhamouda, S. Halevi, and T. Halevi. Supporting private data on hyperledger fabric with secure multiparty computation. *IBM Journal of Research and Development*, 63(2/3): 3:1–3:8, 2019. 15

BIBLIOGRAPHY

[27] Fabrice Benhamouda, Shai Halevi, and Tzipora Halevi. Supporting private data on hyperledger fabric with secure multiparty computation. In *2018 IEEE International Conference on Cloud Engineering, IC2E*. IEEE Computer Society, 2018. doi: 10.1109/IC2E.2018.00069. 52

[28] Karthikeyan Bhargavan, Antoine Delignat-Lavaud, Cédric Fournet, Anitha Gollamudi, Georges Gonthier, Nadim Kobeissi, Natalia Kulatova, Aseem Rastogi, Thomas Sibut-Pinote, Nikhil Swamy, et al. Formal verification of smart contracts: Short paper. In *Proceedings of the 2016 ACM Workshop on Programming Languages and Analysis for Security*, pages 91–96, 2016. 15

[29] IBM Blockchain. Marbles chaincode, 2021. URL https://github.com/IBM-Blockchain-Archive/marbles. 72, 73

[30] David Brumley and Dawn Song. Privtrans: Automatically partitioning programs for privilege separation. In *USENIX Security Symposium*, volume 57, 2004. 13

[31] Frank Cangialosi, Neil Agarwal, Venkat Arun, Srinivas Narayana, Anand Sarwate, and Ravi Netravali. Privid: Practical, Privacy-Preserving video analytics queries. In *USENIX Symposium on Networked Systems Design and Implementation*, April 2022. ISBN 978-1-939133-27-4. 5, 12, 85

[32] M. Castro and B. Liskov. Practical byzantine fault tolerance. In *OSDI*. USENIX Association, 1999. 56

[33] Miguel Castro, Barbara Liskov, et al. Practical byzantine fault tolerance. In *OSDI*, volume 99, pages 173–186, 1999. 52

[34] Ethan Cecchetti, Fan Zhang, Yan Ji, Ahmed Kosba, Ari Juels, and Elaine Shi. Solidus: Confidential distributed ledger transactions via pvorm. In *ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2017. 15, 52

[35] AI CITY CHALLENGE. Ai city challenge dataset 2021, August 2021. https://www.aicitychallenge.org/2021-data-and-evaluation/ (accessed on 10 March 2023). xii, 36, 50, 92, 95

[36] David Chaum, Claude Crépeau, and Ivan Damgard. Multiparty unconditionally secure protocols. In *Proceedings of the 20th annual ACM symposium on Theory of computing*, pages 11–19, 1988. 89

BIBLIOGRAPHY

[37] CoreOS. Clair: Automatic container vulnerability and security scanning for appc and docker, 2021. URL https://coreos.com/clair/docs/latest/. 14

[38] Victor Costan and Srinivas Devadas. Intel sgx explained. *Cryptology ePrint Archive*, 2016. 16, 83, 90

[39] Ronald Cramer, Ivan Damgård, and Ueli Maurer. General secure multi-party computation from any linear secret-sharing scheme. In *International Conference on the Theory and Applications of Cryptographic Techniques*, pages 316–334. Springer, 2000. 80, 89

[40] Ronald Cramer, Ivan Damgård, and Ueli Maurer. General secure multi-party computation from any linear secret-sharing scheme. In *International Conference on the Theory and Applications of Cryptographic Techniques*, pages 316–334. Springer, 2000. 82

[41] Ronald Cramer, Ivan Bjerre Damgård, et al. *Secure multiparty computation*. Cambridge University Press, 2015. 16

[42] Ronald Cramer, Ivan Bjerre Damgård, et al. *Secure multiparty computation*. Cambridge University Press, 2015. 78, 79, 80

[43] Ronald Cramer, Ivan Damgård, Daniel Escudero, Peter Scholl, and Chaoping Xing. SPDZ2k: Efficient MPC mod 2^k for dishonest majority, 2018. 89

[44] Brian Curran. What are oracles? smart contracts, chainlink and the oracle problem, September 2018. https://blockonomi.com/oracles-guide/ (accessed on 23 October 2023). 26

[45] Gaby G. Dagher, Benedikt Bünz, Joseph Bonneau, Jeremy Clark, and Dan Boneh. Provisions: Privacy-preserving proofs of solvency for bitcoin exchanges. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security (CCS)*. ACM, 2015. doi: 10.1145/2810103.2813674. 52

[46] Michael del Castillo. The dao attacked: Code issue leads to $60 million ether theft. 2016. 14

[47] Uriel Feige, Amos Fiat, and Adi Shamir. Zero-knowledge proofs of identity. In *Journal of Cryptology*, pages 1432–1378, New York, NY, 1988. Springer New York. doi: 10.1007/BF02351717. URL https://doi.org/10.1007/BF02351717. 80

BIBLIOGRAPHY

[48] Adrienne Porter Felt, Matthew Finifter, Joel Weinberger, and David Wagner. Diesel: Applying privilege separation to database access. In *Proceedings of the 6th ACM symposium on information, computer and communications security*, pages 416–422, 2011. 13

[49] Marta Fernandez, Paula Moral, Alvaro Garcia-Martin, and Jose M. Martinez. Vehicle re-identification based on ensembling deep learning features including a synthetic training dataset, orientation and background features, and camera verification. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, June 2021. viii, xi, 36, 37, 86, 92

[50] M. Flickner, H. Sawhney, W. Niblack, J. Ashley, Qian Huang, B. Dom, M. Gorkani, J. Hafner, D. Lee, D. Petkovic, D. Steele, and P. Yanker. Query by image and video content: the qbic system. *Computer*, 28(9):23–32, 1995. doi: 10.1109/2.410146. 12

[51] Joel Frank, Cornelius Aschermann, and Thorsten Holz. {ETHBMC}: A bounded model checker for smart contracts. In *29th {USENIX} Security Symposium ({USENIX} Security 20)*, 2020. 15

[52] Xing Gao, Zhongshu Gu, Mehmet Kayaalp, Dimitrios Pendarakis, and Haining Wang. Containerleaks: Emerging security threats of information leakages in container clouds. In *2017 47th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pages 237–248. IEEE, 2017. 14

[53] Rosario Gennaro, Craig Gentry, and Bryan Parno. Non-interactive verifiable computing: Outsourcing computation to untrusted workers. In *Advances in Cryptology–CRYPTO 2010: 30th Annual Cryptology Conference, Santa Barbara, August, 2010.*, pages 465–482. Springer, 2010. 82

[54] Craig Gentry. Fully homomorphic encryption using ideal lattices. In *Proceedings of the Forty-First Annual ACM Symposium on Theory of Computing*, STOC '09, page 169–178, New York, NY, USA, 2009. Association for Computing Machinery. ISBN 9781605585062. doi: 10.1145/1536414.1536440. URL https://doi.org/10.1145/1536414.1536440. 80

[55] Mariana-Iuliana Georgescu, Antonio Bărbălău, Radu Tudor Ionescu, Fahad Shahbaz Khan, Marius Claudiu Popescu, and Mubarak Shah. Anomaly detection in video via self-supervised and multi-task learning. *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 12737–12747, 2020. 13

BIBLIOGRAPHY

[56] Seyedhamed Ghavamnia, Tapti Palit, Azzedine Benameur, and Michalis Polychronakis. Confine: Automated system call policy generation for container attack surface reduction. In *23rd International Symposium on Research in Attacks, Intrusions and Defenses ({RAID} 2020)*, pages 443–458, 2020. 14

[57] Oded Goldreich. Secure multi-party computation. *Manuscript. Preliminary version*, 78 (110):1–108, 1998. 79, 80, 82

[58] S Goldwasser, S Micali, and C Rackoff. The knowledge complexity of interactive proof-systems. In *Proceedings of the Seventeenth Annual ACM Symposium on Theory of Computing*, STOC '85, 1985. ISBN 0897911512. 81, 82

[59] Vipul Goyal, Hanjun Li, Rafail Ostrovsky, Antigoni Polychroniadou, and Yifan Song. Atlas: efficient and scalable mpc in the honest majority setting. In *Advances in Cryptology–CRYPTO 2021: 41st Annual International Cryptology Conference, CRYPTO 2021*. Springer, 2021. 89

[60] Hongpeng Guo, Shuochao Yao, Zhe Yang, Qian Zhou, and Klara Nahrstedt. Crossroi: Cross-camera region of interest optimization for efficient real time video analytics at scale. In *12th ACM Multimedia Systems Conference*, MMSys '21. Association for Computing Machinery, 2021. ISBN 9781450384346. 13

[61] HackerOne. Hyperledger bug bounty program. https://hackerone.com/hyperledger/hacktivity?type=team. 60

[62] Charles Herrmann, Chen Wang, Richard Strong Bowen, Emil Keyder, and Ramin Zabih. Object-centered image stitching. In *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018. 86, 87

[63] Chu-Hong Hoi, Wei Wang, and Michael R Lyu. A novel scheme for video similarity detection. In *Image and Video Retrieval: Second International Conference, CIVR 2003 Urbana-Champaign, IL, USA, July 24–25, 2003 Proceedings 2*, pages 373–382. Springer, 2003. 86, 87

[64] Tyler Hunt, Congzheng Song, Reza Shokri, Vitaly Shmatikov, and Emmett Witchel. Chiron: Privacy-preserving machine learning as a service. *arXiv preprint arXiv:1803.05961*, 2018. 16

[65] Hyperledger. Hyperledger caliper, 2021. URL https://www.hyperledger.org/use/caliper. 72

BIBLIOGRAPHY

[66] Hyperledger. Hyperledger fabric - github, 2023. URL https://github.com/hyperledger/fabric. 36

[67] Intel. Linux-sgx, 2024. URL https://github.com/intel/linux-sgx. 92

[68] Gorka Irazoqui, Mehmet Sinan Inci, Thomas Eisenbarth, and Berk Sunar. Wait a minute! a fast, cross-vm attack on aes. In *International Workshop on Recent Advances in Intrusion Detection*, pages 299–319. Springer, 2014. 14

[69] Istio. Istio, 2021. URL https://istio.io/latest/. 54, 57, 65, 72

[70] Samvit Jain, Xun Zhang, Yuhao Zhou, Ganesh Ananthanarayanan, Junchen Jiang, Yuanchao Shu, Paramvir Bahl, and Joseph Gonzalez. Spatula: Efficient cross-camera video analytics on large camera networks. In *2020 IEEE/ACM Symposium on Edge Computing (SEC)*, pages 110–124, 2020. 13

[71] Facundo Lezama Javier Couto. A guide to video analytics: Applications and opportunities, August 2022. https://tryolabs.com/guides/video-analytics-guide (accessed on 23 October 2023). 1

[72] Junchen Jiang, Ganesh Ananthanarayanan, Peter Bodik, Siddhartha Sen, and Ion Stoica. Chameleon: Scalable adaptation of video analytics. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*, SIGCOMM '18, page 253–266, New York, NY, USA, 2018. Association for Computing Machinery. ISBN 9781450355674. doi: 10.1145/3230543.3230574. 13

[73] Arun Joseph. Perqs code. https://gitlab.com/arunjoseph0/perqs-code, 2026. Version 1.0. 50

[74] Arun Joseph, Nikita Yadav, Vinod Ganapathy, and Dushyant Behl. A contributory public-event recording and querying system. In *Proceedings of the Eighth ACM/IEEE Symposium on Edge Computing*, SEC '23, page 185–198, New York, USA, 2024. Association for Computing Machinery. ISBN 9798400701238. 85, 92

[75] Manu Joseph, Harsh Raj, Anubhav Yadav, and Aaryamann Sharma. Askyourdb: An end-to-end system for querying and visualizing relational databases using natural language, 2022. 12

[76] Sukrit Kalra, Seep Goel, Mohan Dhawan, and Subodh Sharma. Zeus: Analyzing safety of smart contracts. In *NDSS*, 2018. 15, 52

BIBLIOGRAPHY

[77] Daniel Kang, Peter Bailis, and Matei Zaharia. Blazeit: optimizing declarative aggregation and limit queries for neural network-based video analytics. *arXiv preprint arXiv:1805.01046*, 2018. 5, 12, 18, 27, 48

[78] Daniel Kang, Francisco Romero, Peter D. Bailis, Christos Kozyrakis, and Matei Zaharia. VIVA: an end-to-end system for interactive video analytics. In *12th Conference on Innovative Data Systems Research, CIDR 2022, Chaminade, CA, USA, January 9-12, 2022*. www.cidrdb.org, 2022. URL https://www.cidrdb.org/cidr2022/papers/p75-kang.pdf. 5, 12

[79] Andrej Karpathy, George Toderici, Sanketh Shetty, Thomas Leung, Rahul Sukthankar, and Li Fei-Fei. Large-scale video classification with convolutional neural networks. *2014 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1725–1732, 2014. 13

[80] Jonathan Katz and Yehuda Lindell. *Introduction to modern cryptography: principles and protocols*. Chapman and hall/CRC, 2007. 79, 81

[81] Marcel Keller. MP-SPDZ: A versatile framework for multi-party computation. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, 2020. xi, 89, 92, 93

[82] Marcel Keller and Avishay Yanai. Efficient maliciously secure multiparty computation for ram. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 91–124. Springer, 2018. 89

[83] Marcel Keller, Emmanuela Orsini, and Peter Scholl. Mascot: faster malicious arithmetic secure computation with oblivious transfer. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 830–842, 2016. 89

[84] A. Khochare, A. Krishnan, and Y. Simmhan. A scalable platform for distributed object tracking across a many-camera network. *IEEE Transactions on Parallel Distributed Systems*, 32(06):1479–1493, jun 2021. ISSN 1558-2183. doi: 10.1109/TPDS.2021.3049450. 26

[85] Tong Kong, Liming Wang, Duohe Ma, Zhen Xu, Qian Yang, and Kai Chen. A secure container deployment strategy by genetic algorithm to defend against co-resident attacks in cloud computing. In *2019 IEEE 21st International Conference on High Performance Computing and Communications; IEEE 17th International Conference on Smart*

BIBLIOGRAPHY

City; IEEE 5th International Conference on Data Science and Systems (HPCC/SmartCity/DSS), pages 1825–1832. IEEE, 2019. 14

[86] Jay Kreps, Neha Narkhede, Jun Rao, et al. Kafka: A distributed messaging system for log processing. In *Proceedings of the NetDB*, volume 11, pages 1–7, 2011. 52

[87] Maxwell Krohn. Building secure high-performance web services with OKWS. In *USENIX Annual Technical Conference*, 2004. 13

[88] Kubernetes. Kubernetes golang api, 2021. https://godoc.org/golang.org/x/build/kubernetes/api. 65

[89] Kubernetes. Kubernetes: Production grade container orchestration, 2023. URL https://kubernetes.io/. 36, 54, 57, 65

[90] Kai Lei, Qichao Zhang, Limei Xu, and Zhuyun Qi. Reputation-based byzantine fault-tolerance for consortium blockchain. In *2018 IEEE 24th International Conference on Parallel and Distributed Systems (ICPADS)*, pages 604–611. IEEE, 2018. 52

[91] Tianye Li, Mira Slavcheva, Michael Zollhoefer, Simon Green, Christoph Lassner, Changil Kim, Tanner Schmidt, Steven Lovegrove, Michael Goesele, Richard Newcombe, et al. Neural 3d video synthesis from multi-view video. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5521–5531, 2022. 86, 87

[92] Xiaoxiao Li and Chen Change Loy. Video object segmentation with joint re-identification and attention-aware mask propagation. In *Proceedings of the European conference on computer vision (ECCV)*, pages 90–105, 2018. 86

[93] Yuanqi Li, Arthi Padmanabhan, Pengzhan Zhao, Yufei Wang, Guoqing Harry Xu, and Ravi Netravali. Reducto: On-camera filtering for resource-efficient real-time video analytics. In *SIGCOMM '20*. ACM, 2020. ISBN 9781450379557. 13

[94] Xin Lin, Lingguang Lei, Yuewu Wang, Jiwu Jing, Kun Sun, and Quan Zhou. A measurement study on linux container security: Attacks and countermeasures. In *Proceedings of the 34th Annual Computer Security Applications Conference*, pages 418–429, 2018. 14

[95] Yehida Lindell. Secure multiparty computation for privacy preserving data mining. In *Encyclopedia of Data Warehousing and Mining*, pages 1005–1009. IGI global, 2005. 82

BIBLIOGRAPHY

[96] Yehuda Lindell. Secure multiparty computation. *Communications of the ACM*, 64(1), 2020. 89

[97] Yehuda Lindell and Ariel Nof. A framework for constructing fast mpc over arithmetic circuits with malicious adversaries and an honest-majority. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 259–276, 2017. 89

[98] Yehuda Lindell and Benny Pinkas. Privacy preserving data mining. In *Annual international cryptology conference*, pages 36–54. Springer, 2000. 82

[99] Yehuda Lindell and Benny Pinkas. An efficient protocol for secure two-party computation in the presence of malicious adversaries. In *Advances in Cryptology-EUROCRYPT 2007: 26th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Barcelona, Spain, May 20-24, 2007. Proceedings 26*, pages 52–78. Springer, 2007. 81

[100] Peng Liu, Bozhao Qi, and Suman Banerjee. Edgeeye: An edge service framework for real-time intelligent video analytics. In *EdgeSys'18*. Association for Computing Machinery, 2018. ISBN 9781450358378. 13

[101] Xinchen Liu, Wu Liu, Tao Mei, and Huadong Ma. A deep learning-based approach to progressive vehicle re-identification for urban surveillance. In *Computer Vision–ECCV 2016: 14th European Conference, Amsterdam, October 2016, Proceedings*. Springer, 2016. 86

[102] Loi Luu, Duc-Hiep Chu, Hrishi Olickel, Prateek Saxena, and Aquinas Hobor. Making smart contracts smarter. In *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*, pages 254–269, 2016. 15, 52

[103] Filipe Manco, Costin Lupu, Florian Schmidt, Jose Mendes, Simon Kuenzer, Sumit Sati, Kenichi Yasukata, Costin Raiciu, and Felipe Huici. My vm is lighter (and safer) than your container. In *Proceedings of the 26th Symposium on Operating Systems Principles*, pages 218–233, 2017. 14

[104] Easwar Vivek Mangipudi, Krutarth Rao, Jeremy Clark, and Aniket Kate. Towards automatically penalizing multimedia breaches (extended abstract). In *2019 IEEE European Symposium on Security and Privacy Workshops, EuroS&P Workshops 2019, Stockholm, Sweden, June 17-19, 2019*, 2019. doi: 10.1109/EuroSPW.2019.00044. 57

BIBLIOGRAPHY

[105] Sujit Biswas Milind Naphade, Vinay Kolar. Multi-camera large-scale intelligent video analytics with deepstream sdk, November 2018. https://developer.nvidia.com/blog/multi-camera-large-scale-iva-deepstream-sdk (accessed on 23 October 2023). 1

[106] D.L. Mills. Internet time synchronization: the network time protocol. *IEEE Transactions on Communications*, 39(10):1482–1493, 1991. doi: 10.1109/26.103043. 27

[107] Payman Mohassel and Peter Rindal. Aby3: A mixed protocol framework for machine learning. In *Proceedings of the 2018 ACM SIGSAC conference on computer and communications security*, pages 35–52, 2018. 16

[108] Payman Mohassel and Yupeng Zhang. Secureml: A system for scalable privacy-preserving machine learning. In *2017 IEEE symposium on security and privacy (SP)*, pages 19–38. IEEE, 2017. 15

[109] Soo-Jin Moon, Vyas Sekar, and Michael K Reiter. Nomad: Mitigating arbitrary cloud side channels via provider-assisted migration. In *Proceedings of the 22nd acm sigsac conference on computer and communications security*, pages 1595–1606, 2015. 14

[110] Antonio Muñoz, Ruben Rios, Rodrigo Román, and Javier López. A survey on the (in) security of trusted execution environments. *Computers & Security*, 129:103180, 2023. 78, 83

[111] Jaehyun Nam, Seungsoo Lee, Hyunmin Seo, Phil Porras, Vinod Yegneswaran, and Seungwon Shin. {BASTION}: A security enforcement network stack for container networks. In *2020 {USENIX} Annual Technical Conference ({USENIX}{ATC} 20)*, pages 81–95, 2020. 14

[112] Neha Narula, Willy Vasquez, and Madars Virza. zkledger: Privacy-preserving auditing for distributed ledgers. In *Usenix Symposium on Networked Systems Design and Implementation (NSDI)*, 2018. 15

[113] Valeria Nikolaenko, Udi Weinsberg, Stratis Ioannidis, Marc Joye, Dan Boneh, and Nina Taft. Privacy-preserving ridge regression on hundreds of millions of records. In *2013 IEEE symposium on security and privacy*, pages 334–348. IEEE, 2013. 15

[114] Ivica Nikolić, Aashish Kolluri, Ilya Sergey, Prateek Saxena, and Aquinas Hobor. Finding the greedy, prodigal, and suicidal contracts at scale. In *Proceedings of the 34th Annual Computer Security Applications Conference*, pages 653–663, 2018. 15

BIBLIOGRAPHY

[115] Thomas L. Norman. Chapter 6 - electronics elements: A detailed discussion. In Lawrence J. Fennelly, editor, *Effective Physical Security (Fifth Edition)*, pages 95–137. Butterworth-Heinemann, 2017. ISBN 978-0-12-804462-9. doi: https://doi.org/10.1016/B978-0-12-804462-9.00006-3. URL https://www.sciencedirect.com/science/article/pii/B9780128044629000063. 1

[116] Olga Ohrimenko, Felix Schuster, Cédric Fournet, Aastha Mehta, Sebastian Nowozin, Kapil Vaswani, and Manuel Costa. Oblivious {Multi-Party} machine learning on trusted processors. In *25th USENIX Security Symposium (USENIX Security 16)*, pages 619–636, 2016. 16

[117] D. Ongaro. *Consensus: Bridging theory and practice*. PhD thesis, Stanford University, 2014. 56

[118] Diego Ongaro and John Ousterhout. In search of an understandable consensus algorithm. In *2014 {USENIX} Annual Technical Conference ({USENIX}{ATC} 14)*, pages 305–319, 2014. 52

[119] Ubuntu LXC Package. Lxc sysrawio abuse, 2015. URL https://bugs.launchpad.net/ubuntu/+source/lxc/+bug/1511197/. 13, 60

[120] Paul Pearce, Adrienne Porter Felt, Gabriel Nunez, and David Wagner. Addroid: Privilege separation for applications and advertisers in android. In *Proceedings of the 7th ACM Symposium on Information, Computer and Communications Security*, pages 71–72, 2012. 13

[121] Rishabh Poddar, Ganesh Ananthanarayanan, Srinath Setty, Stavros Volos, and Raluca Ada Popa. Visor: Privacy-Preserving video analytics as a cloud service. In *29th USENIX Security Symposium (USENIX Security 20)*, pages 1039–1056. USENIX Association, August 2020. ISBN 978-1-939133-17-5. 5, 13, 85

[122] Andrew Poelstra, Adam Back, Mark Friedenbach, Gregory Maxwell, and Pieter Wuille. Confidential assets. In *International Conference on Financial Cryptography and Data Security*, pages 43–63. Springer, 2018. 15, 52

[123] Alex Poms, Will Crichton, Pat Hanrahan, and Kayvon Fatahalian. Scanner: Efficient video analysis at scale. *ACM Trans. Graph.*, 37(4), jul 2018. ISSN 0730-0301. doi: 10.1145/3197517.3201394. URL https://doi.org/10.1145/3197517.3201394. 5, 12

BIBLIOGRAPHY

[124] Jordi Pont-Tuset, Federico Perazzi, Sergi Caelles, Pablo Arbeláez, Alex Sorkine-Hornung, and Luc Van Gool. The 2017 davis challenge on video object segmentation. *arXiv preprint arXiv:1704.00675*, 2017. 86

[125] Jean-Jacques Quisquater, Myriam Quisquater, Muriel Quisquater, Michaël Quisquater, Louis Guillou, Marie Annick Guillou, Gaïd Guillou, Anna Guillou, Gwenolé Guillou, and Soazig Guillou. How to explain zero-knowledge protocols to your children. In Gilles Brassard, editor, *Advances in Cryptology — CRYPTO' 89 Proceedings*, pages 628–631, New York, NY, 1990. Springer New York. ISBN 978-0-387-34805-6. 80

[126] Mark Raasveldt and Hannes Mühleisen. Duckdb: an embeddable analytical database. In *Proceedings of the 2019 International Conference on Management of Data*, pages 1981–1984, 2019. URL https://doi.org. 48

[127] Ravi Kiran Raman, Roman Vaculín, Michael Hind, Sekou L. Remy, Eleftheria Kyriaki Pissadaki, Nelson Kibichii Bore, Roozbeh Daneshvar, Biplav Srivastava, and Kush R. Varshney. Trusted multi-party computation and verifiable simulations: A scalable blockchain approach. *arXiv*, 2018. URL http://arxiv.org/abs/1809.08438. 15

[128] Joseph Redmon and Ali Farhadi. Yolov3: An incremental improvement. *ArXiv*, abs/1804.02767, 2018. 12, 87

[129] E. Regnath, N. Shivaraman, S. Shreejith, A. Easwaran, and S. Steinhorst. Blockchain, what time is it? trustless datetime synchronization for iot. In *2020 International Conference on Omni-layer Intelligent Systems (COINS)*, 2020. doi: 10.1109/COINS49042.2020.9191420. 26

[130] relax E. Abebe et al. Enabling enterprise blockchain interoperability with trusted data transfer (industry track). In *Middleware Conference Industrial Track*, 2019. 52

[131] Shaoqing Ren, Kaiming He, Ross B. Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39:1137–1149, 2015. 13

[132] MB Sandler, L Hayat, and LDF Costa. Benchmarking processors for image processing. *Microprocessors and Microsystems*, 1990. ISSN 0141-9331. 91, 92

[133] Clara Schneidewind, Ilya Grishchenko, Markus Scherer, and Matteo Maffei. ethor: Practical and provably sound static analysis of ethereum smart contracts. *arXiv preprint arXiv:2005.06227*, 2020. 15

BIBLIOGRAPHY

[134] Steven M Seitz, Brian Curless, James Diebel, Daniel Scharstein, and Richard Szeliski. A comparison and evaluation of multi-view stereo reconstruction algorithms. In *2006 IEEE computer society conference on computer vision and pattern recognition (CVPR'06)*, volume 1, pages 519–528. IEEE, 2006. 87

[135] AMD Sev-Snp. Strengthening vm isolation with integrity protection and more. *White Paper, January*, 53:1450–1465, 2020. 83

[136] Graham Shaw. Fabric security assessment management report, September 2017. https://wiki.hyperledger.org/download/attachments/13861997/management_report_linux_foundation_fabric_august_2017_v1.1.pdf. 60

[137] Reza Shokri and Vitaly Shmatikov. Privacy-preserving deep learning. In *Proceedings of the 22nd ACM SIGSAC conference on computer and communications security*, pages 1310–1321, 2015. 15

[138] Rui Shu, Xiaohui Gu, and William Enck. A study of security vulnerabilities on docker hub. In *Proceedings of the Seventh ACM on Conference on Data and Application Security and Privacy*, pages 269–280, 2017. 14

[139] Karen Simonyan and Andrew Zisserman. Two-stream convolutional networks for action recognition in videos. In *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'14, Cambridge, MA, USA, 2014. 13

[140] Bharat Singh, Tim K. Marks, Michael J. Jones, Oncel Tuzel, and Ming Shao. A multi-stream bi-directional recurrent neural network for fine-grained action detection. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1961–1970, 2016. 13

[141] Solidity. Solidity, 2021. URL https://docs.soliditylang.org/. 15

[142] Saleem Durai M. A. Sreenu G. Intelligent video surveillance: a review through deep learning techniques for crowd analysis. In *Journal of Big Data 6*. 2019. doi: https://doi.org/10.1186/s40537-019-0212-5. URL https://www.sciencedirect.com/science/article/pii/B9780128044629000063. 1

[143] Nancy Sumrall and Manny Novoa. Trusted computing group (tcg) and the tpm 1.2 specification. In *Intel Developer Forum*, volume 32. Intel Santa Clara, CA, USA, 2003. 78, 83

BIBLIOGRAPHY

[144] Yuqiong Sun, David Safford, Mimi Zohar, Dimitrios Pendarakis, Zhongshu Gu, and Trent Jaeger. Security namespace: making linux security frameworks available to containers. In *27th {USENIX} Security Symposium ({USENIX} Security 18)*, pages 1423–1439, 2018. 14

[145] Ramana Sundararaman, Cedric De Almeida Braga, Eric Marchand, and Julien Pettre. Tracking pedestrian heads in dense crowd. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 3865–3875, 2021. 86

[146] Byungchul Tak, Canturk Isci, Sastry Duri, Nilton Bila, Shripad Nadgowda, and James Doran. Understanding security implications of using containers in the cloud. In *2017 {USENIX} Annual Technical Conference ({USENIX}{ATC} 17)*, pages 313–319, 2017. 14

[147] The PostgreSQL Global Development Group. Postgresql. https://www.postgresql.org/, 2026. Version 18.4. 48

[148] Tradelens. Tradelens: Digitizing the global supply chain, 2021. 59

[149] D. Tran, L. Bourdev, R. Fergus, L. Torresani, and M. Paluri. Learning spatiotemporal features with 3d convolutional networks. In *2015 IEEE International Conference on Computer Vision (ICCV)*, Los Alamitos, CA, USA, dec 2015. 13

[150] Stacey Truex, Nathalie Baracaldo, Ali Anwar, Thomas Steinke, Heiko Ludwig, Rui Zhang, and Yi Zhou. A hybrid approach to privacy-preserving federated learning. In *Proceedings of the 12th ACM workshop on artificial intelligence and security*, pages 1–11, 2019. 16

[151] Chia-Che Tsai, Donald E Porter, and Mona Vij. {Graphene-SGX}: A practical library {OS} for unmodified applications on {SGX}. In *2017 USENIX Annual Technical Conference*, 2017. 92

[152] Chia-Che Tsai, Donald E Porter, and Mona Vij. Gramine-sgx, 2024. URL https://github.com/gramineproject/gramine. 92

[153] Ultralytics. Yolov3 in pytorch ¿ onnx ¿ coreml ¿ tflite, 2023. URL https://github.com/ultralytics/yolov3. 37, 44

[154] Marten van Dijk, Craig Gentry, Shai Halevi, and Vinod Vaikuntanathan. Fully homomorphic encryption over the integers. In Henri Gilbert, editor, *Advances in Cryptology –*

BIBLIOGRAPHY

EUROCRYPT 2010, pages 24–43, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg. ISBN 978-3-642-13190-5. 80

[155] Mark Patel Vasanth Ganesan, Yubing Ji. Video meets the internet of things, December 2016. https://www.mckinsey.com/industries/technology-media-and-telecommunications/our-insights/video-meets-the-internet-of-things (accessed on 23 October 2023). 1

[156] Paul Voigtlaender, Jonathon Luiten, Philip HS Torr, and Bastian Leibe. Siam r-cnn: Visual tracking by re-detection. In *IEEE/CVF conference on computer vision and pattern recognition*, 2020. 86

[157] Duc-Quang Vu, Ngan T. H. Le, and Jia-Ching Wang. Self-supervised learning via multi-transformation classification for action recognition. *ArXiv*, abs/2102.10378, 2021. 13

[158] Mitre Common Vulnerabilities and Exposures. Cve-2014-6407, 2014. URL http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2014-6407. 13, 60

[159] Mitre Common Vulnerabilities and Exposures. Cve-2015-3631, 2015. URL http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2015-3631. 13, 60

[160] Mitre Common Vulnerabilities and Exposures. Cve-2015-3627, 2015. URL http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2015-3627. 13, 60

[161] Mitre Common Vulnerabilities and Exposures. Cve-2014-9357, 2021. URL http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2014-9357. 13, 60

[162] Dan Wang, Xinrui Cui, Xun Chen, Zhengxia Zou, Tianyang Shi, Septimiu Salcudean, Z Jane Wang, and Rabab Ward. Multi-view 3d reconstruction with transformers. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 5722–5731, 2021. 86, 87

[163] Han Wang, Shangyu Xie, and Yuan Hong. Videodp: A flexible platform for video analytics with differential privacy. *Proceedings on Privacy Enhancing Technologies*, 2020:277 – 296, 2020. 5

[164] Junjue Wang, Brandon Amos, Anupam Das, Padmanabhan Pillai, Norman Sadeh, and Mahadev Satyanarayanan. A scalable and privacy-aware iot service for live video analytics. In *MMSys'17*. Association for Computing Machinery, 2017. ISBN 9781450350020. 5, 85

BIBLIOGRAPHY

[165] Limin Wang, Yuanjun Xiong, Zhe Wang, Yu Qiao, Dahua Lin, Xiaoou Tang, and Luc Van Gool. Temporal segment networks: Towards good practices for deep action recognition. In *European Conference on Computer Vision*, 2016. 13

[166] Rui Wang, Yixue Hao, Yiming Miao, Long Hu, and Min Chen. Rt3c: Real-time crowd counting in multi-scene video streams via cloud-edge-device collaboration. *IEEE Transactions on Services Computing*, 2024. 86, 87

[167] X. Wang, Ross B. Girshick, Abhinav Kumar Gupta, and Kaiming He. Non-local neural networks. *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2017. 13

[168] Yuya Watanabe, Teruhisa Hochin, and Hiroki Nomiya. Method of similarity retrieval of color videos based on impressions. In *2016 IEEE/ACIS 15th International Conference on Computer and Information Science (ICIS)*, pages 1–6. IEEE, 2016. 86, 87

[169] Danielle Whittaker. Why ai cctv is the future of security and surveillance in public spaces, December 2021. https://www.securitymagazine.com/articles/96719-why-ai-cctv-is-the-future-of-security-and-surveillance-in-public-spaces (accessed on 23 October 2023). 1

[170] Zhirong Wu, Shuran Song, Aditya Khosla, Fisher Yu, Linguang Zhang, Xiaoou Tang, and Jianxiong Xiao. 3d shapenets: A deep representation for volumetric shapes. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1912–1920, 2015. 86

[171] Tiantu Xu, Luis Materon Botelho, and Felix Xiaozhu Lin. Vstore: A data store for analytics on large videos. In *Proceedings of the Fourteenth EuroSys Conference 2019*, EuroSys '19, New York, NY, USA, 2019. Association for Computing Machinery. ISBN 9781450362818. doi: 10.1145/3302424.3303971. URL https://doi.org/10.1145/3302424.3303971. 5, 12

[172] Andrew C. Yao. Protocols for secure computations. In *23rd Annual Symposium on Foundations of Computer Science (sfcs 1982)*, pages 160–164, 1982. 78, 79, 80, 82, 89

[173] Yuval Yarom and Katrina Falkner. Flush+ reload: a high resolution, low noise, l3 cache side-channel attack. In *23rd {USENIX} Security Symposium ({USENIX} Security 14)*, pages 719–732, 2014. 14

BIBLIOGRAPHY

[174] Juheon Yi, Utku Günay Acer, Fahim Kawsar, and Chulhong Min. Argus: Enabling cross-camera collaboration for video analytics on distributed smart cameras. *IEEE Transactions on Mobile Computing*, 2024. 86, 87

[175] Xiaoyi Yu, Kenta Chinomi, Takashi Koshimizu, Naoko Nitta, Yoshimichi Ito, and Noboru Babaguchi. Privacy protecting visual processing for secure video surveillance. In *2008 15th IEEE International Conference on Image Processing*, pages 1672–1675, 2008. doi: 10.1109/ICIP.2008.4712094. 5

[176] Samee Zahur, Mike Rosulek, and David Evans. Two halves make a whole: Reducing data transfer in garbled circuits using half gates. In *EUROCRYPT 2015*. Springer, 2015. 89

[177] Dominik Zapletal and Adam Herout. Vehicle re-identification for automatic video traffic surveillance. In *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*, pages 25–31, 2016. 86

[178] Cong Zhang, Hongsheng Li, Xiaogang Wang, and Xiaokang Yang. Cross-scene crowd counting via deep convolutional neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 833–841, 2015. 87

[179] Fan Zhang and Feng Liu. Parallax-tolerant image stitching. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3262–3269, 2014. 86

[180] Haoyu Zhang, Ganesh Ananthanarayanan, Peter Bodik, Matthai Philipose, Paramvir Bahl, and Michael J. Freedman. Live video analytics at scale with approximation and Delay-Tolerance. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*, pages 377–392, Boston, MA, March 2017. USENIX Association. ISBN 978-1-931971-37-9. URL https://www.usenix.org/conference/nsdi17/technical-sessions/presentation/zhang. 5, 12

[181] Mengya Zhang, Xiaokuan Zhang, Yinqian Zhang, and Zhiqiang Lin. {TXSPECTOR}: Uncovering attacks in ethereum from transactions. In *29th {USENIX} Security Symposium ({USENIX} Security 20)*, pages 2775–2792, 2020. 15, 52

[182] Mingwei Zhang, Daniel Marino, and Petros Efstathopoulos. Harbormaster: Policy enforcement for containers. In *2015 IEEE 7th International Conference on Cloud Computing Technology and Science (CloudCom)*, pages 355–362. IEEE, 2015. 14

BIBLIOGRAPHY

[183] Tan Zhang, Aakanksha Chowdhery, Paramvir (Victor) Bahl, Kyle Jamieson, and Suman Banerjee. The design and implementation of a wireless video surveillance system. In *MobiCom '15*. Association for Computing Machinery, 2015. ISBN 9781450336192. 13

[184] Yinqian Zhang, Ari Juels, Michael K Reiter, and Thomas Ristenpart. Cross-tenant side-channel attacks in paas clouds. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, pages 990–1003, 2014. 14

[185] Wolfie Zhao. $30 million: Ether reported stolen due to parity wallet breach. 2017. 14

[186] Hanrui Zhong, Yingpeng Sang, Yongchun Zhang, and Zhicheng Xi. Secure multi-party computation on blockchain: An overview. In Hong Shen and Yingpeng Sang, editors, *Parallel Architectures, Algorithms and Programming*, pages 452–460. Springer, 2020. 15

[187] Bolei Zhou, Agata Lapedriza, Jianxiong Xiao, Antonio Torralba, and Aude Oliva. Learning deep features for scene recognition using places database. *Advances in neural information processing systems*, 2014. 87

[188] Yi Zhou, Mingjun Cao, Jingdi You, Ming Meng, Yuehua Wang, and Zhong Zhou. Mr video fusion: interactive 3d modeling and stitching on wide-baseline videos. In *Proceedings of the 24th ACM Symposium on Virtual Reality Software and Technology*, pages 1–11, 2018. 86, 87

[189] Guy Zyskind, Oz Nathan, and Alex Pentland. Enigma: Decentralized computation platform with guaranteed privacy. *arXiv*, abs/1506.03471, 2015. URL http://arxiv.org/abs/1506.03471. 15