Data Security and Privacy Preservation for Mobile Edge Computing

by

Lingshuang Liu

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Doctor of Philosophy
in
Electrical and Computer Engineering

Waterloo, Ontario, Canada, 2026

© Lingshuang Liu 2026

Examining Committee Membership

The following served on the Examining Committee for this thesis. The decision of the Examining Committee is by majority vote.

External Examiner:	Vojislav Misic Professor Department of Computer Science Toronto Metropolitan University
Supervisor:	Xuemin (Sherman) Shen University Professor Department of Electrical and Computer Engineering University of Waterloo
Internal Member:	Sagar Naik Professor Department of Electrical and Computer Engineering University of Waterloo
Internal Member:	Anwar Hasan Professor Electrical and Computer Engineering University of Waterloo
Internal-External Member:	Jun Liu Professor Department of Applied Mathematics University of Waterloo

Author's Declaration

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

Mobile Edge Computing (MEC) is an emerging network architecture that extends cloud computing capabilities to the network edge, enabling low-latency, bandwidth-efficient, and context-aware services. By processing and storing data closer to end devices, MEC effectively alleviates the burden on centralized cloud infrastructures and supports latency-sensitive and data-intensive applications such as autonomous driving, health monitoring, and real-time video analytics. However, the distributed, heterogeneous, and resource-constrained nature of MEC introduces substantial security and privacy challenges. Edge servers may be semi-trusted or compromised, end devices can behave maliciously, and external adversaries may corrupt the sensitive data throughout their lifecycle, from service access to data storage and processing, threatening confidentiality, integrity, and availability.

This thesis aims to systematically address these challenges by designing secure, efficient, and privacy-preserving mechanisms tailored to MEC environments. Focusing on the entire lifecycle of edge data, we develop secure and efficient cryptographic solutions that protect service access, data storage, and distributed processing, while respecting the practical constraints of edge computing. The main contributions of this thesis are summarized as follows.

First, we propose E-DAC, an efficient and distributed service access control framework for MEC. E-DAC enables fine-grained service authorization, service access control, and mutual authentication between edge servers and end devices without relying on centralized authorities. By extending key-aggregate cryptosystems, E-DAC allows service providers to issue constant-size aggregate authorization keys for multiple services, significantly reducing key management overhead and improving the efficiency in service delegation. In addition, E-DAC leverages secret sharing to enable flexible and efficient user authorization for service access at the network edge, and supports zero round-trip authentication between users and edge servers for reducing communication overhead. These properties make E-DAC particularly suitable for real-time MEC applications that demand frequent interactions, low latency, and strong security guarantees, such as intelligent surveillance systems, smart transportation, and location-based edge services.

Second, we propose SecureEdge, an efficient and secure data integrity verification framework for MEC in data storage. By leveraging lightweight cryptographic mechanisms, SecureEdge enables both private and public verifiability, allowing data owners and third-party auditors (TPAs) to verify the integrity of edge-stored data using efficient cryptographic constructions in the multiplicative and bilinear groups, respectively. SecureEdge also supports secure multi-replica storage, allowing user data to be distributed across multiple edge

servers to enhance data availability and fault tolerance. It further prevents collusion attacks by generating distinct randomized replicas for different edge nodes while preserving a unified authentication tag for proof generation. This design ensures integrity, privacy, and efficiency simultaneously, preventing TPAs from inferring sensitive data during public auditing. SecureEdge is therefore well suited for practical MEC deployments requiring transparent and trustworthy data storage.

Third, we propose SVTAgg, a secure, verifiable, and traceable model aggregation protocol for MEC in federated learning based data processing. Unlike conventional federated learning architectures that assume trustworthy aggregation servers, SVTAgg explicitly addresses the privacy and integrity risks posed by untrusted edge servers which are responsible for intermediate aggregation in edge-assisted federated learning. Specifically, to preserve the confidentiality of local gradients, SVTAgg combines secret sharing with a pairwise double masking mechanism that prevents information leakage while still supporting efficient model aggregation. To guarantee the correctness of aggregated results, SVTAgg incorporates homomorphic verifiable tags, enabling lightweight verification of intermediate model updates without revealing client data. Furthermore, SVTAgg introduces client traceability by integrating homomorphic signatures with proxy re-signing signatures, providing authenticated provenance of submitted gradients and enabling the identification and attribution of end devices participating in local model training. SVTAgg establishes a robust aggregation foundation for trustworthy federated learning at the network edge, enabling secure and privacy-preserving collaboration in emerging applications such as autonomous systems, mobile health, and large-scale IoT analytics.

Overall, this thesis advances the state of the art in MEC security by providing a unified set of efficient and scalable solutions that secure service access, data storage, and distributed processing across the edge data lifecycle. By explicitly addressing the unique threats and constraints of MEC environments, this work enables the safe and trustworthy deployment of critical applications, including cyber-physical systems, mobile health services, and collaborative edge intelligence. The proposed frameworks lay a solid foundation for future secure edge computing systems and contribute to the broader vision of privacy-preserving, resilient, and trustworthy edge-enabled infrastructures.

Acknowledgements

The past four years of my PhD study at the University of Waterloo have been among the most unique, precious, and rewarding periods of my life. I am deeply grateful to all those who have supported me throughout this journey. In particular, I would like to thank my supervisor, thesis committee members, colleagues, and family. Without their guidance, encouragement, and unwavering support, I would not have been able to achieve the research outcomes presented in this thesis or fully enjoy my PhD experience.

First and foremost, I would like to express my sincere and heartfelt gratitude to my supervisor, Professor Xuemin (Sherman) Shen. His patient guidance, insightful advice, and constant encouragement have been instrumental in the successful completion of this thesis. Beyond helping me develop strong academic and research skills, Professor Shen consistently inspired me to pursue excellence and maintain high professional standards. What I appreciate most is his exceptional understanding and support, both academically and personally. I am truly inspired by his dedication to research, his students, and his family, as well as his enthusiasm and integrity as a scholar and mentor. I would also like to thank Professor Vojislav Misic for serving as the external examiner of my thesis and for sharing his valuable insights into network security. I am equally grateful to the honorable members of my thesis committee, Professor Jun Liu, Professor Sagar Naik, and Professor Anwar Hasan. Their thoughtful comments, constructive suggestions, and careful evaluations have significantly strengthened both the technical substance and the presentation of this work.

While research can often be challenging and demanding, my colleagues in the BBCR group made this journey both enjoyable and memorable. I would like to especially thank Professor Weihua Zhuang, Professor Cheng Huang, Professor Dongxiao Liu, Professor Lang Xue, Professor Dan Zhu, Professor Conghao Zhou, Dr. Shisheng Hu, Dr. Xinyu Huang, Dr. Yingying Pei, Dr. Mingcheng He, and Xue Qin for their inspiring discussions and invaluable insights related to my research. I am sincerely grateful to all BBCR group members for their continuous encouragement, generous help, and the many meaningful moments we shared together. There are many others whose names are not explicitly mentioned here. This in no way diminishes my appreciation for their support and kindness. I feel truly privileged to have worked with and shared this journey alongside so many talented, energetic, and supportive individuals.

Finally, I would like to express my gratitude to my husband, Dr. Jianbing Ni, and my two beloved sons, Ricky Ni and Ryan Ni. Their love, understanding, and encouragement have been, and will always be, my greatest source of inspiration. I am deeply thankful for

their constant care and support, which made me feel their presence even during the most challenging times. I will continue to work hard toward my career goals and strive to make them proud.

Dedication

This PhD thesis is dedicated to my beloved mother, Mingzhen Yu.

Table of Contents

Examining Committee ii

Author's Declaration iii

Abstract iv

Acknowledgements vi

Dedication viii

List of Figures xiii

List of Tables xiv

List of Abbreviations xv

1 Introduction 1

1.1 Mobile Edge Computing (MEC) 1

1.1.1 Mobile Edge Computing (MEC) Architecture 2

1.1.2 MEC Applications 4

1.1.3 Key Functions of Edge Nodes in MEC 5

1.2 Security and Privacy in MEC 6

1.3 Motivations and Objectives 9

1.4 Research Contributions 9
1.5 Thesis Outline 11

2 Background and Literature Review 12
2.1 Mathematical Background 12
2.2 Technical Primitives 13
2.2.1 Pairwise Double Masking. 13
2.2.2 Secret Sharing 15
2.2.3 Key Agreement 16
2.2.4 Zero-Round-Trip Communication 16
2.3 Literature Review 17
2.3.1 Secure Access Control in Edge Computing 17
2.3.2 Edge Data Integrity Verification 18
2.3.3 Secure Model Aggregation in Federated Learning (FL) 20
2.4 Summary 23

3 E-DAC: Efficient and Distributed Access Control for Mobile Edge Computing 24
3.1 Introduction 24
3.2 Models and Design Goals 27
3.2.1 System Model 27
3.2.2 Security Model 29
3.2.3 Design Goals 34
3.3 E-DAC 34
3.3.1 Overview of E-DAC 34
3.3.2 Detailed E-DAC 35
3.4 Security Analysis 40
3.5 Performance Evaluation 45
3.6 Summary 50

4 SecureEdge: Secure Data Integrity Verification for Mobile Edge Computing 51

4.1 Introduction . . . 51

4.2 Models and Design Goals . . . 53

4.2.1 System Model . . . 53

4.2.2 Security Model . . . 57

4.2.3 Design Goals . . . 59

4.3 SecureEdge . . . 60

4.3.1 Overview of SecureEdge . . . 60

4.3.2 Detailed SecureEdge . . . 61

4.3.3 Data Dynamics . . . 66

4.4 Security Analysis . . . 71

4.5 Performance Evaluation . . . 75

4.6 Summary . . . 78

5 SVTAgg: Secure, Verifiable, and Traceable Federated Learning for Mobile Edge Computing 79

5.1 Introduction . . . 79

5.2 Models and Design Goals . . . 82

5.2.1 System Model . . . 82

5.2.2 Threat Model . . . 83

5.2.3 Design Goals . . . 85

5.3 SVTAgg . . . 85

5.3.1 Overview of SVTAgg . . . 85

5.3.2 Detailed SVTAgg . . . 87

5.4 Security Analysis . . . 93

5.5 Performance Evaluation . . . 99

5.5.1 Experimental Setup . . . 100

5.5.2 Model Accuracy 100
5.5.3 Computational Overhead Analysis 102
5.5.4 Communication Efficiency 106
5.6 Summary 107

6 Conclusion and Future Works **108**
6.1 Main Research Contributions 108
6.2 Future Research Directions 110

References **113**

List of Figures

1.1 General Architecture of MEC 3

3.1 System Model 27
3.2 Computational Overhead Comparison of an Efficient and Distributed Service Access Control Framework (E-DAC), APECS, and AADEC 47
3.3 Communication Overhead Comparison of E-DAC, APECS, and AADEC . 49

4.1 System Model 54
4.2 Example of Merkle Hash Tree (MHT) and Sequence-enforced MHT (sMHT) 68
4.3 Example of sMHT Update under Block Modification Operation 69
4.4 Example of sMHT Update under Block Insert Operation 70
4.5 Example of sMHT Update under Block Deletion Operation 71
4.6 Computational Overhead Comparison of a Secure Edge Data Integrity Verification Framework (SecureEdge) and Existing Schemes 76
4.7 Comparison between Private and Public Verification and Batch Verification 77

5.1 System Architecture of a Secure, Verifiable, and Traceable Model Aggregation Protocol (SVTAgg) 84
5.2 Model Accuracy with SVTAgg and without SVTAgg. 101
5.3 Model Accuracy and Computational Cost of Model Training of SVTAgg. . 102
5.4 Computational Cost of Different Parties in SVTAgg. 103
5.5 Computational Cost of Cloud Server with Client Drop-out. 105
5.6 Comparison on Computational Cost of Cloud Server. 106

List of Tables

3.1 Notations of E-DAC 36
3.2 Performance of E-DAC 46
4.1 Notations of SecureEdge 62
5.1 Notations of SVTAgg 88

List of Abbreviations

0-RTT Zero-Round Trip 26, 28, 33, 38, 50

AAI Auxiliary Authentication Information 66, 69–71

ABE Attribute-Based Encryption 17, 18, 25, 49

AR/VR Augmented Reality and Virtual Reality 4, 5, 50, 54

BLS Boneh–Lynn–Shacham 20

CDH Computational Diffie-Hellman 12, 16, 41–43

CONF Conference-Key Sharing 13, 97, 98

d-BDHE decisional-Bilinear Diffie–Hellman Exponent 13, 44, 45

DBDH Decisional Bilinear Diffie Hellman 13, 44, 45

DDOS Distributed Denial-of-Service 7

DHI Diffie-Hellman Inversion 12, 13, 95–97

DOS Denial-of-Service 6

E-DAC an Efficient and Distributed Service Access Control Framework xiii, xiv, 10, 12, 13, 25–27, 34, 35, 40–50, 108, 109

EFL Edge-assisted Federated Learning 10–12, 21–23, 79–86, 92, 100, 101, 104–107, 109

FFT Fast Fourier Transform 20

FL Federated Learning x, 13, 14, 17, 20–23, 79, 80, 83, 107, 111, 112

IID Independent and Identically Distributed 21, 22, 100, 111

IoT Internet of Things 1, 2, 4, 5, 7, 17, 19, 50, 79, 107, 111

IoV Internet of Vehicles 21

KAC Key-Aggregate Cryptosystem 26, 34

MAC Message Authentication Code 10, 52, 61, 64, 71–73, 75, 85

MEC Mobile Edge Computing ix, 1, 2, 4–10, 17–19, 21, 24–30, 32, 33, 50–53, 56, 57, 78, 108–112

MHT Merkle Hash Tree xiii, 66, 68

PoR Proof of Retrievability 52, 60

PPT Probabilistic Polynomial-Time 95–97

PRF Pseudo-Random Function 61, 63, 75

PRP Psudo-Ramdom Permutation 61, 63, 75

SecureAgg Secure Aggregation 81, 106

SecureEdge a Secure Edge Data Integrity Verification Framework xiii, 10, 12, 13, 52, 53, 59–61, 66, 68, 71, 73–78, 109

SGD Stochastic Gradient Descent 83, 100

sMHT Sequence-enforced MHT xiii, 66, 68–71

SVTAgg a Secure, Verifiable, and Traceable Model Aggregation Protocol xiii, 10–13, 81–87, 92, 93, 95, 99–107, 109–111

TEE Trusted Execution Environment 21, 22

TPA Third-Party Auditor 10, 19, 52, 53, 55, 58–60, 63–66, 73, 74, 77, 78, 109

Chapter 1

Introduction

1.1 Mobile Edge Computing (MEC)

Nowadays, massive volumes of data are being generated on mobile and Internet of Things (IoT) devices, placing significant demands on computational, communication, and storage resources for data transmission, processing, and management. This surge in resource consumption poses serious challenges to traditional cloud computing architectures. The geographical separation between data generation at the edge and data processing in remote cloud servers introduces substantial transmission overhead and bandwidth consumption [1]. It creates significant barriers to deploying applications that require low-latency responses and high-bandwidth data handling. MEC is a distributed computing paradigm that brings computation and data storage closer to the location where it is needed, particularly near mobile users and IoT devices at the network edge [2]. Instead of sending all data to centralized cloud centers for processing, MEC allows computation to be performed at the edge nodes, such as cellular base stations, gateways, or other edge servers.

MEC significantly enhances traditional cloud computing by decentralizing computation and storage, bringing them closer to end users at the network edge. This proximity drastically reduces communication latency, making MEC ideal for latency-sensitive applications such as autonomous driving, augmented and virtual reality, and smart manufacturing systems [3]. Unlike centralized cloud servers, MEC nodes can access rich local contextual information, such as device locations, user mobility patterns, and real-time network conditions, enabling dynamic, context-aware resource allocation and faster service delivery. This localized intelligence empowers MEC to meet the stringent responsiveness, reliability,

and bandwidth efficiency demands of emerging edge applications, which cloud computing alone cannot effectively fulfill.

1.1.1 MEC Architecture

The general architecture of MEC, as shown in Figure 1.1, is structured as a three-layer hierarchy comprising the device layer, edge layer, and cloud layer, organized in increasing order of storage and computing capabilities. A variety of communication technologies are employed to connect entities within and across layers, including wired technologies, such as cable, optical fiber, and Ethernet, and wireless technologies, including 4G/5G, Wi-Fi, Bluetooth, and satellite links [4].

The device layer consists of a wide range of end devices owned by the users, broadly classified into IoT devices and mobile devices. IoT devices, such as sensors, radio-frequency identification tags, smart home systems, wearables (e.g., fitness trackers), and industrial IoT components, typically run lightweight operating systems with limited computational power and storage. The users often deploy these devices in static environments for specific tasks, such as obstacle detection, temperature monitoring, and fitness tracking. Due to their resource limitations, IoT devices primarily collect raw data, perform minimal processing, and offload tasks to the upper layers for further computation. Mobile devices, such as smartphones, tablets, smartwatches, and smart vehicles, are more portable and equipped with complex operating systems (e.g., Android, iOS, Windows) [5]. Compared to IoT devices, they support high processing capabilities and larger storage, enabling the execution of complex applications like gaming, video streaming, and multitasking.

The edge layer adopts a hierarchical structure consisting of multiple sublayers of edge servers (or edge nodes) situated between end devices and cloud servers. These edge servers are arranged with increasing computational power from the lower to the upper sublayers. The servers in the lowest sublayer typically include various network access points, such as wireless base stations, routers, and embedded servers in devices (e.g., smartphones, tablets). They receive data collected by end devices and initially process tasks based on their available computing power. If a task exceeds their capacity, it is offloaded to the upper sublayer edge servers with greater computational capabilities. Content caching follows a similar hierarchical model. Additionally, edge servers across different sublayers are responsible for relaying control instructions from the cloud servers down the hierarchy. The edge layer extends cloud computing to the edge of the network by pushing computation and storage resources closer to end users. This not only alleviates the computing and storage burden on resource-constrained end devices but also enables support for real-time

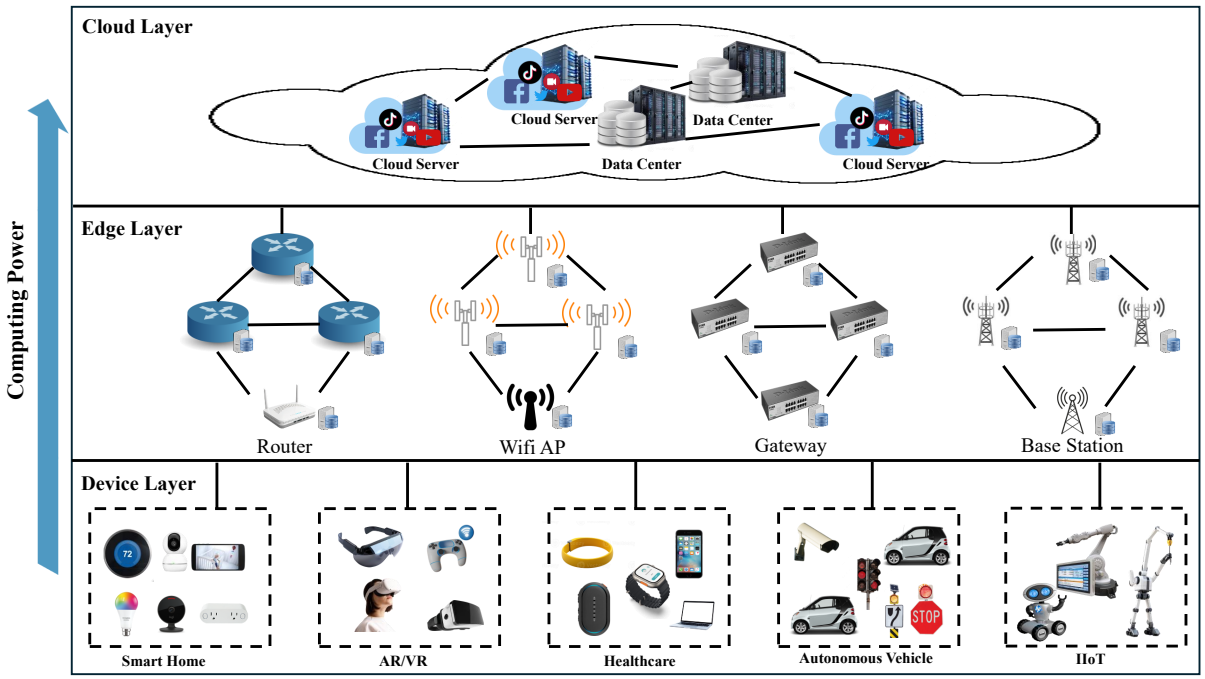


Figure 1.1: General Architecture of MEC

and latency-sensitive applications (e.g., vehicle navigation [6], online gaming, industrial automation [7]) by offloading such services from the cloud to edge servers. Moreover, because edge servers are deployed in close proximity to users, they can capture contextual information, such as location and mobility patterns, more promptly, enabling effective context-aware computing and services.

The cloud layer hosts cloud servers and data centers. It aggregates vast computing and storage resources to provide various network services to mobile subscribers. The service providers deploy on-demand services on these servers which are accessible to users anytime and anywhere with an Internet connection. This layer leverages virtualization technologies to create isolated virtual machines for different applications, allowing simultaneous, independent service provisioning for multiple users. During service delivery, cloud servers receive massive volumes of data from edge servers, including raw data collected by end devices and intermediate data pre-processed at the edge. This allows cloud servers to analyze information and make decisions from a global perspective. Additionally, the cloud layer plays a supervisory role, establishing policies to guide edge servers in executing delegated services, especially latency-sensitive applications.

1.1.2 MEC Applications

MEC brings computing and storage resources closer to end users, enabling a wide range of applications that benefit from low latency, real-time data processing, and distributed storage capabilities. MEC applications can be broadly categorized into three major types: real-time services, IoT applications, and storage and caching services.

Real-Time Services. MEC significantly minimizes data transmission time and reduces service response latency, which allow MEC to support real-time data processing and control, making it well-suited for latency-sensitive applications. For example, MEC plays a pivotal role in road data acquisition and decision-making for autonomous driving [8]. MEC infrastructure, such as roadside units, cameras, and sensors, is strategically deployed along roadways to support vehicle-to-vehicle, vehicle-to-infrastructure, and vehicle-to-pedestrian communications, gathering diverse traffic data and processing it in close proximity to moving vehicles. This enables real-time analysis and decision-making, allowing autonomous vehicles to respond immediately by adjusting speed, changing lanes, or avoiding obstacles and pedestrians to ensure safety. MEC is also widely regarded as a key enabler for Augmented Reality and Virtual Reality (AR/VR) applications [9]. By offloading computationally intensive tasks to nearby edge servers rather than distant cloud data centers, MEC significantly reduces communication latency and ensures rapid feedback, maintaining synchronization between user actions and virtual interactions. In addition, MEC supports a wide range of other latency-sensitive applications, including online gaming [10], remote surgery [11], and robotics [12], where low delay and reliable responsiveness are vital.

IoT Applications. MEC can efficiently process and analyze large volumes of IoT data by bringing computing and storage resources closer to the data source. This proximity not only reduces the latency associated with data processing but also alleviates the bandwidth burden on the backhaul network. Such capabilities are particularly valuable for data-intensive IoT applications, enabling them to make intelligent decisions and implement effective strategies based on real-time big data analytics. For example, in industrial IoT [13], a large number of sensors and IoT devices are deployed across factory environments to monitor the operations and status of industrial equipment. These devices are interconnected via a network and continuously generate substantial volumes of industrial data. With MEC, industrial data can be processed and analyzed locally at the network edge in real time. This capability supports predictive maintenance by enabling the timely detection of potential equipment failures and the optimization of repair schedules, ultimately reducing operational costs and enhancing overall system performance. Additionally, MEC

can improve the quality of telemedicine [14] by supporting low-latency video consultations and real-time diagnostics, thereby expanding access to healthcare services, especially in underserved or rural areas. Other IoT-enabled applications, such as smart cities [15], environmental monitoring [16], and smart grid systems [17], also utilize MEC for low-latency data processing, real-time decision-making, and efficient resource management.

Storage and Caching Services. MEC optimizes data storage by bringing storage resources closer to both data sources and end users at the network edge. This localized storage approach alleviates the burden on end devices, particularly IoT devices with limited storage capacity, by enabling them to offload generated data to nearby edge servers. Moreover, storing data at the edge ensures faster access to frequently requested content, reducing the reliance on distant cloud servers and thereby improving response times and overall system efficiency. For example, edge servers can deliver content caching services for frequently accessed web content, including videos, images, and other web data [3]. By proactively caching popular content at the network edge, MEC allows end users to retrieve data from nearby edge servers instead of remote cloud servers. This significantly reduces content delivery latency and enhances the overall user experience. In addition, by storing replicas of important data at the network edge, MEC enables rapid data recovery in the event of device failures or network attacks, thereby enhancing system fault tolerance [18]. Offloading data to nearby edge servers allows resource-constrained end devices to conserve local storage by safely deleting the original copies.

1.1.3 Key Functions of Edge Nodes in MEC

The MEC service architecture introduces edge nodes for service access, data storage, and data processing, fundamentally reshaping the entire data lifecycle.

Service Access on Edge Nodes. MEC enables users and devices to access a wide range of services directly at the network edge, significantly shortening data transmission distance. This proximity reduces service response latency and enhances the performance of latency-sensitive applications such as AR/VR and online gaming, thereby improving the overall user experience. MEC platforms also allow for dynamic deployment of services to edge nodes based on user demand and real-time network conditions. This flexibility enables operators to efficiently scale services up or down, optimize resource utilization, and maintain high-quality service delivery.

Data Storage on Edge Nodes. Data storage on edge nodes is a vital service of MEC, enabling data generated by end devices to be stored locally at the network edge.

This proximity significantly reduces retrieval latency, making MEC ideal for data-intensive and latency-sensitive applications. Moreover, storing data at the edge alleviates bandwidth pressure on core and backhaul networks, making it well-suited for high-throughput scenarios like video surveillance and streaming. Edge servers typically employ intelligent storage mechanisms, such as caching frequently accessed content to minimize response times. For instance, video streaming services can cache trending videos at the edge to reduce buffering and enhance user experience. Advanced storage techniques, such as data compression, deduplication, and encryption, are also integrated to maximize storage efficiency and safeguard sensitive information. In addition, MEC supports hierarchical storage management, where critical or time-sensitive data is retained locally for immediate access, while less urgent data is offloaded to the cloud for long-term storage. This layered strategy not only improves system responsiveness and efficiency but also enhances privacy and security by reducing the exposure of sensitive data during transmission.

Data Processing on Edge Nodes. By performing data processing at the network edge, MEC minimizes the communication delay associated with transmitting data to remote cloud servers, enabling real-time data analysis and decision-making. This is especially critical for latency-sensitive applications. MEC also supports the deployment of AI and machine learning models at the edge to perform advanced tasks like image recognition, video analysis, and real-time inference with minimal delay. Moreover, MEC facilitates collaborative intelligence by enabling distributed edge nodes to share computational workloads. A prominent example is federated learning, where multiple edge devices collaboratively train a shared AI model without exposing their raw data. This not only enhances efficiency and scalability but also strengthens data privacy by keeping sensitive information localized.

1.2 Security and Privacy in MEC

In MEC, edge nodes are usually deployed by third-party service providers or network vendors, they are typically considered honest-but-curious: while they follow application protocols and do not actively alter data for malicious purposes, they may attempt to infer sensitive user information, such as identity, location, or personal interests, from the data they process. Moreover, the deployment of MEC expands the attack surface by placing edge nodes in less physically secure environments, making them attractive targets for cyberattacks. Compared to cloud servers, edge nodes typically have more limited computational and storage resources, which restricts their ability to implement robust and complex security mechanisms. This resource limitation increases their vulnerability to a range of threats, including Denial-of-Service (DOS) attacks, data breaches, and malware

infections. We analyze the security and privacy threats in MEC for the core functions of edge nodes.

1) Security and Privacy Challenges in Edge-based Service Access

Edge servers are delegated by cloud service providers to offer localized services to end users within their coverage areas. Typically, user registration is handled by centralized cloud servers, while service access is managed by the edge servers. This separation between registration and service access introduces additional overhead in managing secure service access, particularly regarding authentication and authorization. As such, enforcing access control and preventing unauthorized users from accessing services on edge nodes becomes a critical concern in MEC environments. One of the primary threats is identity and credential forgery. Malicious actors can forge identities or spoof service access credentials to gain illegitimate access to edge services. Once successful, these attackers may initiate large-scale Distributed Denial-of-Service (DDOS) attacks by sending numerous service requests to the targeted edge node, overwhelming it with traffic and disrupting service delivery.

However, efficient access control in MEC is particularly challenging due to resource limitations of edge nodes and high mobility of end devices. Specifically, edge nodes typically lack the robust computational, storage, and energy resources available in centralized cloud environments, so implementing advanced security mechanisms, such as strong cryptographic protocols and fine-grained access control policies, can impose significant burdens on edge infrastructure and degrade service performance. Meanwhile, in MEC scenarios, user devices (e.g., smartphones, autonomous vehicles) are often mobile and maintain only transient connections with edge nodes. Enforcing authentication and access control within these short-lived sessions can consume precious network resources, increasing the likelihood that legitimate service access requests fail due to delays or communication interruptions.

2) Security and Privacy Challenges in Edge-based Data Storage

IoT and mobile devices can outsource the raw data they generate or collect to edge servers to conserve local storage resources. Similarly, service providers or application vendors utilize edge storage capabilities to cache service data, thereby supporting low-latency access for end users. However, once data is offloaded to edge nodes, it falls outside the direct control of the data owners. This poses a significant privacy risk, as the stored data may contain sensitive information, such as patient health records in remote healthcare monitoring, that can be exposed to curious or semi-trusted edge nodes. To preserve data privacy,

robust encryption algorithms are commonly employed. However, encrypted data presents a major challenge for verifying data integrity, as operations on encrypted content are often computationally intensive and non-trivial. Additionally, hardware/software failures or operational errors on edge nodes may result in data loss or unavailability. Due to their limited defense capabilities compared to centralized cloud servers, edge nodes are also more vulnerable to external attacks aimed at tampering with or deleting stored data. Moreover, because edge nodes typically have constrained storage capacity, they may intentionally delete infrequently accessed data in favor of storing new data for financial gain. These factors collectively pose significant challenges to maintaining the integrity, availability, and privacy of data stored at the edge. In a mobile and highly distributed MEC environment, data owners often replicate data across multiple geographically distributed edge nodes, introducing further complexity in maintaining data consistency, confidentiality, and integrity.

3) Security and Privacy Challenges in Edge-based Data Processing

When end users upload raw data in plaintext to edge nodes for processing, sensitive information, such as medical records or financial data, can be exposed to honest-but-curious edge nodes or become vulnerable to unauthorized access by external attackers. Even if users apply pre-processing to the data before uploading, privacy risks persist. Curious edge nodes can potentially infer original user data from the preprocessed input. For instance, in a MEC-based federated learning scenario, users train local models on private data and upload the resulting model parameters to edge nodes for aggregation into a global model. However, edge nodes can infer sensitive training data from local model updates by analyzing the parameters, thereby compromising user privacy [19].

The trustworthiness of computation results is also a critical concern in MEC. When users offload computation-intensive tasks, such as model aggregation or real-time driving decisions, to edge nodes, malicious or self-interested nodes may return incorrect results. For example, in an autonomous driving prediction task, a dishonest edge node might replace the original algorithm with a simplified version to save computation resources, sacrificing accuracy and potentially endangering safety. Similarly, in edge-assisted federated learning, an edge node could manipulate the aggregation process to bias the global model toward its own objectives, compromising the model's utility and fairness [20].

1.3 Motivations and Objectives

This thesis aims to investigate data security and privacy preservation in the MEC framework, focusing on key functions enabled by the edge nodes in the data life cycle at the network edge, namely service access, data storage, and data processing.

1) Secure Access Control for MEC Services. We will study secure and efficient access control in the MEC environment that achieves the following goals: i) Secure authorization: Secure authorization comprises service authorization, in which the service provider delegates the services to the edge servers, and service access authorization, in which the end users register at the service provider to obtain the authorization of service access. ii) Mutual authentication achieves authorization verification for both edge servers and end users. The mutual authentication should be efficient to ensure that the verification could complete in short-lived connection for resource-restricted mobile devices.

2) Secure Data Integrity Verification in MEC. We will investigate secure data integrity verification for MEC that achieves the following goals: i) Secure data integrity verification: The data owner or a third party should efficiently verify whether the data is correctly maintained on the edge servers, without retrieving the whole data from the edge servers. ii) Secure multi-replica storage: The user data is distributed across multiple edge servers to enhance data availability. Batch verification should be supported on the copies stored on multiple edge servers to further enhance computational efficiency for end users during integrity checks.

3) Secure and Verifiable Edge-assisted Federated Learning. We consider the edge-assisted federated learning as one of the key data processing methods at the network edge and propose new solutions to enhance security and privacy in edge-assisted federated learning. Specifically, we will explore secure and verifiable model aggregation in edge-assisted federated learning that achieves the following goals: i) Secure aggregation: To protect local models from privacy leakage, the local models of end users are protected with random noise against the edge servers, the central server, and other end devices, while supporting the aggregation of the protected local models on the edge servers. ii) To ensure the correctness of the final global model, the verification of secure aggregation operations on the edge servers is offered.

1.4 Research Contributions

In this thesis, we propose novel secure and efficient frameworks that achieve the protection of data security and privacy preservation throughout the data life cycle in MEC.

Specifically, the detailed contributions can be summarized in threefold:

1) We propose E-DAC in the MEC environment, where the edge nodes are delegated to provide latency-sensitive computing services to the nearby end devices on behalf of the cloud service provider. E-DAC addresses the challenge of efficient and distributed service access control, comprising edge service authorization, service access authorization, and mutual authentication between edge servers and edge devices. In doing so, E-DAC first extends a key-aggregate cryptosystem to enable batch service authorization, in which a service provider can aggregate the authorization keys of different services to produce a constant-size aggregate key for an edge server. Second, E-DAC enables users to acquire authorization from the service provider for service access on edge servers by using efficient secret sharing. Third, edge servers and users can authenticate with each other without interacting with a centralized server, while enabling secure zero-round trip communication, so that the service data is protected and the communication bandwidth cost is low. In addition, the service provider is capable of efficiently revoking the authorization of the dropout or compromised edge servers or users in response to the dynamics of the MEC environment. Finally, we prove the security of service access control in E-DAC, including unforgeability of service authorization and confidentiality of service data, and conduct extensive analysis and experiments to demonstrate that E-DAC is highly computational and communication-efficient on service authorization, authentication, and revocation.

2) We propose SecureEdge for MEC. By leveraging lightweight cryptographic mechanisms, such as Message Authentication Code (MAC) and hash functions, SecureEdge provides an efficient platform for verifying if edge servers correctly store and maintain end-user data. It supports both private and public verifiability, enabling data owners and Third-Party Auditor (TPA) to initiate integrity verification in the multiplicative group and the bilinear group, respectively. SecureEdge also offers multi-replica secure storage, allowing user data to be distributed across multiple edge servers to enhance data availability and fault tolerance. To prevent edge nodes from exploiting data replicas stored on other nodes to falsely pass integrity verification, SecureEdge can generate distinct randomized replicas for different edge nodes, while still enabling each replica to support proof generation using the same authentication tag. Meanwhile, this data randomization preserves data privacy, ensuring that TPAs cannot infer any sensitive information during the verification process. Moreover, by integrating a sequence-enforce Merkle Hash Tree, SecureEdge supports data dynamics, allowing secure and efficient verification even during data insertions, deletions, and modifications. Our security analysis and experimental evaluation demonstrate that SecureEdge is secure against various data integrity attacks and achieves high computational efficiency in both authentication tag generation and integrity verification.

3) We propose SVTAgg for Edge-assisted Federated Learning (EFL). Unlike conven-

tional federated learning architectures that assume trustworthy aggregation servers, SVTAgg explicitly addresses the privacy and integrity risks posed by untrusted edge servers which are responsible for intermediate aggregation in EFL. Specifically, to preserve the confidentiality of local gradients, SVTAgg combines secret sharing with a pairwise double masking mechanism that prevents information leakage while still supporting efficient model aggregation. To guarantee the correctness of aggregated results, SVTAgg incorporates homomorphic verifiable tags, enabling lightweight verification of intermediate model updates without revealing client data. Furthermore, SVTAgg introduces client traceability by integrating homomorphic signatures with proxy re-signing signatures, providing authenticated provenance of submitted gradients and enabling the identification and attribution of end devices participating in local model training. Security analysis shows that SVTAgg ensures strong gradient privacy, aggregation correctness, and client traceability against realistic adversarial edge settings. Experimental results demonstrate that SVTAgg can achieve similar convergence performance compared with standard EFL, while maintaining low computational and communication overhead.

1.5 Thesis Outline

The remainder of the thesis is organized as follows: Chapter 2 investigates the background and literature review. In Chapters 3, 4, 5, we present our three research contributions, respectively. Finally, we conclude this thesis and introduce future research directions in Chapter 6.

Chapter 2

Background and Literature Review

In this chapter, we present mathematical background, technical primitives, and the existing works related to service access control, data integrity verification, and secure aggregation in EFL.

2.1 Mathematical Background

In this section, we review negligible probability and mathematical hard assumptions on which our E-DAC, SecureEdge, and SVTAgg are based.

Negligible Function. A function $\mathsf{negl} : \mathbb{N} \rightarrow \mathbb{R}^+$ is called *negligible* if for every positive polynomial $p(\cdot)$, there exists an integer N such that for all $\lambda > N$, $\mathsf{negl}(\lambda) < \frac{1}{p(\lambda)}$.

Negligible Probability. An event is said to occur with *negligible probability* if its probability is upper bounded by a negligible function in the security parameter λ, i.e., $\Pr[\mathsf{Event}] \leq \mathsf{negl}(\lambda)$.

Let G and G_T be two multiplicative cyclic groups with a same prime order p. g is a generator of G. G and G_T satisfy a bilinear mapping $\hat{e} : G \times G \rightarrow G_T$.

Computational Diffie-Hellman (CDH) problem [21]: given $(g, g^a, g^b) \in G$, where $a, b \in Z_p$, compute g^{ab}.

CDH assumption: there is no probabilistic polynomial time algorithm $\mathcal{A}$ that has a non-negligible advantage ϵ in solving the CDH problem such that $\mathbf{Pr}[\mathcal{A}(g, g^a, g^b) = g^{ab}] \geq \epsilon$.

Diffie-Hellman Inversion (DHI) problem [22]: given $g, g^a \in G$, where $a \in Z_p$, compute $g^{\frac{1}{a}} \in G$.

DHI assumption: there is no probabilistic polynomial time algorithm $\mathcal{A}$ that has a non-negligible advantage ϵ in solving the DHI problem such that $\mathbf{Pr}[\mathcal{A}(g, g^a) = g^{\frac{1}{a}}] \geq \epsilon$.

Conference-Key Sharing (CONF) problem [23]: given $g, g^s, g^{sv} \in G$, where $s, v \in Z_p$, compute $\hat{e}(g, g)^v \in G_T$.

CONF assumption: there is no probabilistic polynomial time algorithm $\mathcal{A}$ that has a non-negligible advantage ϵ in solving the CONF problem such that $\mathbf{Pr}[\mathcal{A}(g, g^s, g^{sv}) = \hat{e}(g, g)^v] \geq \epsilon$.

Decisional Bilinear Diffie Hellman (DBDH) problem [24]: given $g, g^a, g^b, g^c \in G$ and a random element $R \in G_T$, decide whether $\hat{e}(g, g)^{abc} = R$.

DBDH assumption: there is no probabilistic polynomial time algorithm $\mathcal{A}$ that has a non-negligible advantage ϵ in solving the DBDH problem such that $|\mathbf{Pr}[\mathcal{A}(g, g^a, g^b, g^c, e(g, g)^{abc}) = 1] - \mathbf{Pr}[\mathcal{A}(g, g^a, g^b, g^c, R) = 1]| \geq \epsilon$.

decisional-Bilinear Diffie–Hellman Exponent (d-BDHE) problem [25]: given a vector of $2n$ elements $h, g, g^{\alpha}, g^{\alpha^2}, \cdots, g^{\alpha^n}, g^{\alpha^{n+2}}, g^{\alpha^{2n}} \in G$ and a random element $R \in G_T$, decide whether $\hat{e}(g^{\alpha^{n+1}}, h) = R$.

d-BDHE assumption: there is no probabilistic polynomial time algorithm $\mathcal{A}$ that has a non-negligible advantage ϵ in solving the decisional d-BDHE problem such that $|\mathbf{Pr}[\mathcal{A}(g, g^{\alpha}, g^{\alpha^2}, \cdots, g^{\alpha^n}, g^{\alpha^{n+2}}, g^{\alpha^{2n}}, \hat{e}(g^{\alpha^{n+1}}, h) = 1] - \mathbf{Pr}[\mathcal{A}(g, g^{\alpha}, g^{\alpha^2}, \cdots, g^{\alpha^n}, g^{\alpha^{n+2}}, g^{\alpha^{2n}}, R) = 1]| \geq \epsilon$.

2.2 Technical Primitives

In this section, we introduce the technical primitives used in E-DAC, SecureEdge, and SVTAgg, including pairwise marking, secret sharing, key agreement, and zero-round-trip communication.

2.2.1 Pairwise Double Masking.

Pairwise masking [26] enables a server to compute the sum of clients' local model updates while preventing the disclosure of any individual update, even in the presence of client dropouts and an honest-but-curious server.

Consider a FL system with a set of clients $\mathcal{U} = \{u_1, \ldots, u_n\}$ and a server that aggregates client updates. Each client u_i holds a local model update vector $\boldsymbol{w}_i \in R^d$. The goal of

secure aggregation is to allow the server to learn only the aggregated update $\sum_{i=1}^{n} \boldsymbol{w}_i$, and nothing else about any individual $\boldsymbol{w}_i$.

Pairwise Mask Generation. For every ordered pair of clients (u_i, u_j) with $i < j$, the two clients establish a shared secret key $k_{i,j}$ using a key agreement protocol (e.g., Diffie–Hellman) as a mask seed. From this shared key, both clients derive a pseudorandom mask vector $\boldsymbol{r}_{i,j} = PRG(k_{i,j})$, where $PRG(\cdot)$ is a secure pseudorandom generator that expands the shared key into a vector of the same dimension as the model update.

Mask Construction. Each client u_i constructs its total mask by summing all pairwise masks it shares with other clients, assigning the pairwise masks with the opposite signs to ensure cancellation during aggregation:

$$\boldsymbol{m}_i = \sum_{i<j} \boldsymbol{r}_{i,j} - \sum_{j<i} \boldsymbol{r}_{j,i}.$$

This signed construction guarantees that for every pair (i, j), the mask $\boldsymbol{r}_{i,j}, i < j$ added by client u_i is exactly canceled by the corresponding subtraction performed by client u_j when all masked updates are summed.

Each client u_i computes its masked update as $\tilde{\boldsymbol{w}}_i = \boldsymbol{w}_i + \boldsymbol{m}_i$, and uploads $\tilde{\boldsymbol{w}}_i$ to the server. Individually, each $\tilde{\boldsymbol{w}}_i$ is information-theoretically hidden by the random mask $\boldsymbol{m}_i$.

Aggregation and Mask Cancellation. Upon receiving the masked updates from all participating clients, the server computes

$$\sum_{i=1}^{n} \tilde{\boldsymbol{w}}_i = \sum_{i=1}^{n} \boldsymbol{w}_i + \sum_{i=1}^{n} \boldsymbol{m}_i.$$

By construction of the pairwise masks, the aggregate mask term satisfies $\sum_{i=1}^{n} \boldsymbol{m}_i = \boldsymbol{0}$, as every $\boldsymbol{r}_{i,j}$ appears once with a positive sign and once with a negative sign. Consequently, the server obtains exactly the desired aggregated update $\sum_i \boldsymbol{w}_i$ while learning nothing about any individual update.

A key challenge in practical FL systems is client dropout. If a client drops out after masks have been generated but before submitting its masked update, the cancellation property no longer holds. To address this, the protocol incorporates an additive secret sharing mechanism: each client secret-shares its pairwise mask seeds among other clients.

In the event of a dropout, the server can reconstruct the missing masks using shares from surviving clients and remove them from the aggregate, thereby restoring correctness without compromising privacy.

However, if the server mistakenly claims a client drops out after the client's masked update arrives, it can reconstruct the client's mask by using secret shares of mask seeds from other surviving clients, subsequently removing the mask and obtaining the client's model update.

Double Mask Construction. To address this new threat, each client u_i chooses a random value γ_i as an additional mask seed and derives its secondary mask $PRG(\gamma_i)$ to double mask its model update. The double masked model update is

$$\tilde{\boldsymbol{w}}_i = \boldsymbol{w}_i + PRG(\gamma_i) + \boldsymbol{m}_i.$$

The client generates and distributes the secret shares of the random mask seed to other clients for the cancellation of the secondary mask during model aggregation.

2.2.2 Secret Sharing

Secret sharing enables a secret to be divided into multiple pieces, called *shares*, which are distributed among a set of participants such that only authorized subsets of participants can reconstruct the secret, while unauthorized subsets learn nothing about it. A widely adopted instantiation is the (t, n)-threshold secret sharing scheme. A secret $s \in F$ (where F is a finite field) is shared among n participants in a way that any subset of at least $t+1$ participants can jointly reconstruct the secret s, while any subset of at most t participants obtains no information about s.

In Shamir's secret sharing scheme, the dealer selects a random polynomial $f(x) = s + a_1 x + a_2 x^2 + \cdots + a_t x^t$, where the coefficients $a_1, \ldots, a_t$ are chosen uniformly at random from F. The secret s is embedded as the constant term of the polynomial. Each participant P_i is assigned a public, nonzero identifier $i \in F$ and receives a share $\mathsf{share}_i = f(i)$, so that all shares are $(i, \mathsf{share}_i)_{i \in n} = S.Share(s, t, n)$.

Given any subset of $t+1$ valid shares $\{(i, f(i))\}$, the secret $s = f(0)$ can be reconstructed using Lagrange interpolation:

$$s = S.Recon(\{(i, \mathsf{share}_i)\}_{i \in I}, t) = \sum_{i \in I} f(i) \cdot \prod_{\substack{j \in I \\ j \neq i}} \frac{j}{j - i},$$

where I denotes the index set of participating shares. Fewer than $t+1$ shares provide no information about s.

2.2.3 Key Agreement

Key agreement enables two or more parties to establish a shared secret key over an insecure communication channel, without prior shared secrets. The established key can subsequently be used to derive cryptographic material, such as encryption keys, message authentication codes, or pseudorandom masks. The most widely deployed key agreement protocol is the Diffie-Hellman key exchange. Let (G, q, g) be a cyclic group of a prime order q with a generator g. Alice selects a random private key $a \in Z_q^*$ and publishes the public key g^a. Given Bob's public key g^b, both parties can compute the shared secret

$$K = KA.aggree(g^a, g^b) = g^{ab}.$$

The security of Diffie-Hellman key exchange relies on the hardness of the CDH Problem.

2.2.4 Zero-Round-Trip Communication

Zero-round-trip (0-RTT) communication is a protocol property that allows a client to transmit application-layer data in its first outbound message without waiting for any response from the server in the current communication session.

Let C and S denote a client and a server, respectively, and let t_0 denote the time at which C initiates a new session. A protocol supports 0-RTT communication if C can transmit application data m at t_0 as

$$C \to S : (\mathsf{Init}, \mathsf{Data}(m)),$$

before receiving any message from S in the current session. Therefore, the number of completed request-response exchanges before the transmission of m is

$$N_{\mathrm{RTT}}^{\text{before-data}} = 0.$$

The receiver may process or respond to the application data only after validating the accompanying session information. In secure communication protocols, 0-RTT typically relies on state established during a previous connection, such as a secret key, session ticket, cached server parameters, or pre-shared key.

2.3 Literature Review

In this section, we review the existing work in secure access control, secure data integrity verification, and secure aggregation in FL.

2.3.1 Secure Access Control in Edge Computing

Role-based access control is a simple and manageable approach to assigning permissions to users based on their role within an organization. Hou *et al.* [27] designed a fine-grained access control mechanism to ensure data security in MEC. The role-based access control mechanism integrates with a dynamic fine-grained trusted user grouping scheme based on attributes to assign roles to users according to their group credibility. However, role-based access control has low scalability and dynamism because permission assigning is needed for each role. Attribute-based access control [28] is a highly adaptable and customizable way to implement access control policies based on a wide range of user attributes, making it suitable for distributed or rapidly changing environments. Li *et al.* [29] proposed a verifiable ciphertext-policy Attribute-Based Encryption (ABE)-based access control scheme for edge computing-assisted IoT. The desirable feature of outsourced decryption is supported to mitigate the computational cost of data users with limited resources. Feng *et al.* [30] designed ABE with parallel outsourced decryption based on Spark and MapReduce to further improve computational efficiency for service access authentication. The extension of attribute-based access control includes the protection of access control policy proposed by Li *et al.* [31], utilizing ciphertext-policy ABE with privacy to hide the access control structure of service access. Additionally, multi-authority access authorization introduced by Zhang *et al.* [32] achieves multi-attribute authorities in generating multi-attribute-based private keys for users based on non-interactive multi-authority ABE. Nevertheless, attribute-based access control is typically static, making it challenging to revoke access for edge servers or users. Li *et al.*'s ABE in [29] can support attribute revocation, which is conducted on the group of users with specific attributes, not designed for particular users.

Blockchain [33, 34] has been integrated into edge computing to enhance reliable user management and secure data access. For instance, blockchain is employed to develop an authentication and authorization scheme, fostering effective trust between users and providing resistance against single points of failure in edge computing [35]. Saha *et al.* [36] proposed a consortium blockchain-enabled access control scheme, achieving mutual authentication among smart devices and gateway nodes, as well as between gateway nodes and respective edge servers. Key management is conducted among edge servers and associated cloud servers to establish secret keys for secure communication. While both works

focus on blockchain-based methods for user identity management, blockchain is also utilized for video sharing service subscriptions in vehicular edge computing [37]. Permissioned blockchain and smart contracts record access policies and subscription events, enabling user self-certification and event traceability. Despite offering a unique method for user identity and service management, blockchain-based access control faces challenges in infrastructure design, user scalability, and effective consensus.

Biometric features [38, 39] have emerged as key elements for authentication and access control. This approach assigns access rights based on specific biometric information, such as fingerprints, face, iris, DNA, or palm prints. Authentication and access control involve two stages: enrollment and verification. Initially, users' biometric templates are enrolled in the system, securely stored. Subsequently, a captured biometric sample is compared with the enrolled template to authenticate the user. While biometric features are unique and effective for authentication, their immutability poses a serious threat if samples are leaked, stored on smart cards, or servers.

Dougherty *et al.* [40] proposed a distributed access control framework, APECS, ensuring access control and data confidentiality for dynamic MEC services. It is based on token-based authorization and multi-authority ABE, enabling secure asynchronous access control without requiring an "always-on" service provider. However, it faces challenges of privacy leakage and inefficiency in service authorization. To address privacy leakage, Zhou *et al.* [41] proposed an anonymous and auditable distributed access control framework based on conditional anonymous authentication and auditable multi-authority ABE. It allows users to enjoy edge computing services anonymously if impeccable but lose identity privacy once behaving maliciously. However, the access control based on ABE, particularly multi-authority ABE is not efficient. Meanwhile, the size of the service authorization message increases linearly with the number of services and base stations.

2.3.2 Edge Data Integrity Verification

Remote data integrity verification in MEC allows entities to efficiently verify the integrity and correctness of data stored on edge nodes from a remote location. This concept is inspired by similar mechanisms in cloud storage, which offer integrity guarantees for users' data stored in centralized cloud data centers. However, many remote integrity verification schemes originally developed for cloud environments [42, 43, 44, 45, 46, 47] cannot work well in MEC because of the distributed feature and resource constraints. New schemes have been introduced to improve verification efficiency in edge-specific scenarios [48, 49, 50], which typically fall into three main categories: private verifiability, public verifiability, and cooperative verifiability.

Private Verifiability. Li *et al.* [51] proposed a lightweight edge data integrity verification scheme, called EDI-V, which introduces a novel data structure called the variable Merkle hash tree that employs sampling to reduce verification complexity while providing probabilistic integrity guarantees. Later, Li *et al.* [52] developed EDI-S, an enhanced scheme based on elliptic curve Schnorr signatures, which leverages the aggregable nature of the signature scheme to support batch verification, improving efficiency when verifying large volumes of edge data replicas distributed across multiple servers. Cui *et al.* [53] addressed the edge data integrity challenge using homomorphic tags, enabling batch verification to reduce computation, albeit with higher communication costs. Islam *et al.* [54] proposed an edge data integrity verification scheme based on adaptive Merkle hash trees tailored to multi-vendor, multi-server edge environments with diverse datasets. However, these existing schemes primarily focus on integrity verification in edge caching scenarios, where service providers cache data on edge nodes and retain control over verifying its integrity. In such settings, data-independent verification is not required, as the verifiers (i.e., service providers) possess the original data and have sufficient computational resources to perform verification efficiently. In contrast, our work addresses a fundamentally different challenge: enabling end users or TPA to verify the integrity of their own data stored on edge servers. This is essential for secure MEC applications, where end devices often lack both the original data copies and the computational capacity to execute heavy verification operations.

Public Verifiability. Privately verifiable schemes cannot ensure transparency on the verification results and typically do not support data sharing. Consequently, publicly verifiable schemes have gained increasing attention. Tong *et al.* [55] addressed edge data integrity concerns for end users by introducing TPAs to assist with verification, thereby reducing computational burden on resource-constrained devices. Their protocol incorporates a private tag retrieval mechanism to protect both data privacy and query pattern privacy from the semi-trusted TPA. However, despite supporting batch verification, its reliance on a provable data possession-based structure results in limited verification efficiency. Yan *et al.* [56] proposed a publicly verifiable framework involving a trusted third party to manage data integrity in intelligent railway systems, addressing the complexity of multidimensional and heterogeneous data generated by large-scale IoT deployments. Seyedi *et al.* [57] developed DIVO, a decentralized public verification scheme that leverages game-theoretic techniques to offload verification tasks from IoT devices to edge servers. This scheme reduces the computational load on data owners while maintaining verifiability. Nevertheless, most publicly verifiable schemes generally remain less efficient than their privately verifiable counterparts on tag generation and proof verification in terms of computational and communication costs.

Cooperative Verifiability. Cooperative verification offers a promising approach to validate cached data on edge nodes through collaboration among multiple servers. Li *et al.* [58] proposed CooperEDI, a distributed consensus-based mechanism that enables edge servers storing data replicas to cooperatively verify data integrity. However, this mechanism assumes all participating servers are honest and does not account for Byzantine behavior. To overcome this limitation, Li *et al.* [59] introduced WatchEdge, a Byzantine-tolerant protocol that leverages Boneh–Lynn–Shacham (BLS) signatures and blockchain technology to facilitate secure and verifiable integrity proofs across untrusted edge servers. Zhao *et al.* [60] further advanced cooperative verification by developing a smart contract-based framework tailored for multi-vendor, multi-server edge environments. Their solution employs a smart inspection algorithm to detect unreliable data replicas and supports discriminative inspection, significantly enhancing both communication and computational efficiency. While cooperative verification offers strong potential for improving integrity verification in distributed edge environments, it typically requires additional infrastructure, such as blockchain systems or trusted inspection mechanisms, to ensure transparency, accountability, and resilience against malicious participants.

2.3.3 Secure Model Aggregation in FL

We review the existing privacy-preserving model aggregation mechanisms in FL and the verifiable model aggregation protocols in FL.

2.3.3.1 Secure Model Aggregation

Secure model aggregation has become a foundational component of FL, enabling collaborative model training while safeguarding user data. The seminal work by Bonawitz *et al.* [26] established the first practical secure aggregation protocol, leveraging pairwise masking and additive secret sharing to protect individual model updates even in the presence of client dropouts and honest-but-curious servers. This work catalyzed extensive follow-up research aimed at improving efficiency, scalability, and robustness. Bell *et al.* [61] optimized the original design to reduce computational overhead, while So *et al.* [62] proposed Turbo-Aggregate, which employs sparse graph coding to reduce interaction complexity from $O(n^2)$ to $O(nlog\ n)$. Building on these ideas, Kadhe *et al.* [63] introduced FastSecAgg, a Fast Fourier Transform (FFT)-based multi-secret-sharing construction that substantially enhances scalability. LightSecAgg [64] further reduces communication overhead by eliminating the need to reconstruct individual masks, and FSSA [65] shortens the protocol

from four rounds to three under the honest-but-curious adversarial model, yielding additional efficiency gains. In addition, to improve communication efficiency, Fereidooni *et al.* [66] presented SAFELearn, a generic design for efficient private FL systems that protects against inference attacks. Notably, SAFELearn only needs 2 rounds of communication in each training iteration without the dependance of a trusted third party.

Recent work of secure model aggregation has significantly extended classical privacy-preserving designs to address malicious adversaries, decentralization, and robustness in FL. Tang *et al.* [67] achieved near-optimal corruption and dropout tolerance under malicious models using a non-colluding server–initiator architecture and extractable, equivocable commitments, while Xu *et al.* [68] proposed TAPFed to defend against inference attacks via threshold functional encryption in multi-aggregator settings. Decentralized and trustworthy aggregation has been explored through architectures such as DeTA [69], which combines parameter-level shuffling with confidential computing, and blockchain- and Trusted Execution Environment (TEE)-based frameworks [70] that ensure aggregation integrity and auditability. Efficiency-oriented designs include multi–private-key homomorphic aggregation without trusted third parties [71], secret-sharing–based fault-tolerant protocols [72], and privacy-preserving aggregation tailored for the Internet of Vehicles (IoV) environments [73]. Beyond confidentiality, recent approaches consider robustness, including PriRoAgg [74] with zero-knowledge proofs for robust aggregation, zero-knowledge–based aggregation integrity under Byzantine attacks [75], and secure aggregation augmented with backdoor detection [76]. In short, these works illustrate the evolution of secure aggregation toward comprehensive security guarantees that jointly address privacy, correctness, robustness, and practical deployment constraints.

Secure model aggregation in EFL has attracted increasing attention due to the presence of untrusted edge servers, heterogeneous resources, and non-Independent and Identically Distributed (IID) data distributions. Shen *et al.* [77] proposed LiPFed, a lightweight privacy-preserving FL framework that employs decentralized secure aggregation for edge networks by integrating blockchain and additive secret sharing, thereby protecting both local and global model privacy while enabling smart-contract–based detection of malicious edge models. Zhao *et al.* [78] introduced a deep reinforcement learning-based framework that optimizes secure aggregation and resource orchestration in hierarchical FL assisted by untrusted MEC servers, improving efficiency and fairness under system constraints. To cope with non-IID data and Byzantine attacks, He *et al.* [79] proposed PPBR, a hybrid design combining condensed local differential privacy and packed linearly homomorphic encryption to achieve robust, privacy-preserving aggregation across edge and cloud layers, while tolerating device and edge-server dropouts. Wang *et al.* [80] developed a secure aggregation framework in semi-asynchronous EFL based on supervised differential privacy

and control variables, mitigating privacy leakage and accuracy degradation under client dropouts. More recently, Wang *et al.* [81] proposed RaSA, a robust and adaptive secure aggregation protocol that jointly addresses non-IID data, system heterogeneity, and malicious edge servers through adaptive weighting and coded computing, further enhancing convergence and resilience in adversarial EFL environments. However, these frameworks primarily concentrate on secure model aggregation in EFL, leaving the issues of verifiable aggregation and client traceability largely unexplored.

2.3.3.2 Verifiable Model Aggregation

Verifiable secure aggregation is crucial for ensuring the integrity and correctness of FL outcomes, especially in the presence of misbehaving servers or unreliable end devices. The representative works include VerifyNet by Xu *et al.* [82] and VeriFL by Guo *et al.* [83], which introduce dual-masking mechanisms and homomorphic commitment schemes to verify aggregation correctness without revealing individual model updates. These studies established the foundational design principles for incorporating verifiability into secure aggregation protocols. LightVeriFL [84] further improves efficiency by employing compact homomorphic hash functions and succinct commitment structures, substantially reducing communication and computation overhead while preserving robustness against client dropouts. In addition, TEE has been explored to enable efficient and practical verifiability. For example, OPSA [85] proposes a one-pass secure aggregation protocol that leverages TEE to streamline aggregation, while ensuring that verifiability remains enforceable outside the enclave. This design reduces dependence on trusted hardware while maintaining strong integrity guarantees for aggregation results.

More recently, higher-assurance constructions have been proposed to offer stronger correctness and auditability guarantees. Pairing-based protocols utilize bilinear maps to verify gradient correctness, albeit at the cost of significant cryptographic overhead. In contrast, Zhang *et al.* [86] introduced a zero-knowledge virtual machine-based architecture that combines homomorphic encryption with non-interactive proofs to verify correct local training, achieving both privacy and verifiability. In parallel, blockchain-assisted approaches have gained traction for their auditability and transparency. For instance, PPVFL [87] integrates homomorphic encryption with ECDSA-based digital signatures to support publicly verifiable aggregation, while VerifBFL [88] further leverages zk-SNARKs and incrementally verifiable computation on blockchain infrastructures to enable end-to-end auditability across FL rounds. VerifyDFL by Wang *et al.* [89] advances verifiability in decentralized FL by enabling clients to locally validate the correctness of others' gradients using zero-knowledge input proofs, while preserving privacy. Similarly, WVFL [90]

introduces a secure weighted aggregation protocol that mitigates the impact of erroneous local models while allowing devices to verify aggregation correctness, with all intermediate values kept encrypted to ensure privacy. VeSAFL [91] decentralizes model updates across edge nodes, reduces dependency on centralized cloud servers, and fortifies system resilience and reliability by leveraging multi-server aggregators for privacy-preserving FL in edge computing environments. To address robustness against adversarial updates in hierarchical settings, SHIELD [92] proposes a poisoning-resistant aggregation mechanism specifically tailored for hierarchical FL, overcoming the limitations of flat FL robust aggregation methods. Complementing these efforts, PriVeriFL [93] explores the observation that not all model parameters leak privacy, leading to low-expansion homomorphic aggregation schemes that jointly achieve privacy and verifiability, with threshold variants to defend against collusion between aggregators and malicious clients. Efficiency-oriented verifiable designs, such as LightVeriFL [84], further reduce communication and computation overhead using constant-length homomorphic hashes and one-shot recovery under dropouts. Finally, Xie *et al.* [94] proposed FLVA, a verifiable aggregation framework that avoids bilinear groups through a three-party key negotiation protocol, ensuring privacy-preserving ciphertext aggregation with reduced cryptographic complexity.

However, existing research primarily focuses on verifiability of aggregated models in conventional FL, while verifiability in EFL remains largely underexplored. Moreover, traceability of participating clients has not been systematically investigated, leaving a critical gap in accountability and forensic analysis under adversarial or incentive-driven EFL settings.

2.4 Summary

In this chapter, we provided the necessary mathematical background and technical primitives, followed by a comprehensive review of existing work on service access control, data integrity verification, and secure aggregation in EFL.

Chapter 3

E-DAC: Efficient and Distributed Access Control for Mobile Edge Computing[1]

3.1 Introduction

MEC has gained significant attention for its ability to bring computation and storage closer to data sources, reducing network bandwidth usage and enabling latency-sensitive services. For example, MEC allows prompt processing of video feed data from surveillance cameras on roads by nearby edge nodes, eliminating the need for transmitting to a remote cloud server and enabling quick response to criminal activities and threats. Similarly, in autonomous driving, MEC facilitates real-time processing of on-board camera footage for immediate lane detection and alerts. In short, MEC enhances traditional centralized applications by enabling localized computation and fast decision-making at the edge, improving overall efficiency and responsiveness.

MEC environments are highly dynamic due to the distributed nature of edge infrastructures. Edge nodes may join, leave, or change their service availability in response to user mobility, resource utilization, or network conditions. This dynamic nature presents the following new and intensified challenges for access control between users and edge servers: How to efficiently authorize an edge server to provide a particular service or a set of services for users, and how to efficiently authorize a user to access a particular service

[1] A version of this chapter has been published in IEEE Transactions on Mobile Computing [95].

offered by the edge server in the highly dynamic MEC environment? Meanwhile, due to the mobility of devices, the short-lived connections between users and edge servers make an involving cloud server in the authentication process overwhelming. Traditional access control frameworks [96, 97] utilized in cloud computing, which rely on an "always-online" server for authentication, are not suitable for the distributed MEC environment due to the significant network latency. Therefore, there is an urgent need for a computation and communication-efficient solution that handles authorization and authentication for both edge servers and users to address these challenges in the distributed scenario. Additionally, considering the dynamic nature of MEC, i.e., both users and edge servers may join, leave or be compromised in the less secure edge environment, quick revocation mechanisms for both users and edge servers must be incorporated into the efficient distributed access control framework to ensure secure and timely control over service access of users and edge servers.

The popular distributed access control methods in edge computing are role-based access control [27, 98] and ABE based access control [99, 100]. These methods allow the service provider to be off-line after authorizing services to the edge servers. However, these access control mechanisms are typically static [98], making them challenging to revoke access for edge servers or users. Consequently, they are not well-suited for the dynamic MEC environment. With the advent of blockchain technology, several blockchain-based decentralized access control mechanisms [101, 102, 103] have been proposed to reduce dependency on cloud servers and mitigate the risk of single points of failure. However, these mechanisms face inherent challenges related to consensus, cost, and scalability [104]. Moreover, most of these solutions primarily focus on securing static content services [105, 106], where edge servers provide users with content allocated by a content service provider. In contrast, MEC aims to deliver real-time computing services by leveraging user input data and returning processing results promptly. To address the requirements of dynamics and low latency in MEC, Dougherty et al. [40] proposed a distributed access control framework called APECS for dynamic MEC services, based on multi-authority ABE. APECS supports service authorization and authentication between users and edge servers without relying on an "always-online" cloud server. It also enables the revocation of edge servers to handle turnover. Nonetheless, in APECS, 1) the size of the service authorization message increases linearly with the number of services and base stations, and 2) revoking a compromised edge server requires re-keying all edge servers under the same base station.

Contributions: We study highly efficient solutions for distributed authorization and authentication and propose E-DAC [95], an efficient and distributed access control framework, designed specifically for dynamic MEC services. It enables revocable service authorization for edge servers and users, while facilitating mutual authentication between them

without relying on an "always-online" centralized authentication server by extending the Key-Aggregate Cryptosystem (KAC) [107] and secret sharing-based broadcast encryption [108]. The main contributions of E-DAC are summarized in four-fold:

- E-DAC enables efficient service authorization for edge servers based on KAC, which supports flexible decryption delegation of any subset of the ciphertext with a constant-size decryption key. We extend KAC to achieve efficient service authorization for registered edge servers. This extension allows the service provider to authorize an edge server to offer a set of services using a scalable aggregate authorization key. By utilizing this key, an edge server can prove to the users that it is authorized to provide the specific service they seek to access. The service authorization is efficient as the size of the service authorization key for an edge server is constant, non-linear with the number of authorized services.

- E-DAC achieves service access authorization for users based on efficient secret sharing. Secret shares of a service key are created by the service provider for service access authorization to users. These shares are distributed to the subscribed users and the edge server to enable the edge server to verify the authorization to users. The users can randomize secret shares and demonstrate the ownership of secret shares to authenticate to the edge server for service access. The service access authorization is efficient on computation because of low computational overhead on secret share generation and reconstruction.

- E-DAC supports efficient mutual authentication between an edge server and a user device. To save the communication rounds between the edge server and the user device, secure Zero-Round Trip (0-RTT) mutual authentication is achieved to integrate service message and authentication request into a single message, which can significantly reduce the communication costs and protect the service data of the authenticated user. The correct decryption of service data with the service key and the aggregate authorization key ensures mutual authentication between the user and the edge server.

- E-DAC provides efficient revocation of an edge server by exposing its associated secret share of the service key. This revocation process incurs no additional cost for other edge servers. Additionally, revoking a user entails adding their identity information to a revocation list, ensuring they are denied access to the MEC services.

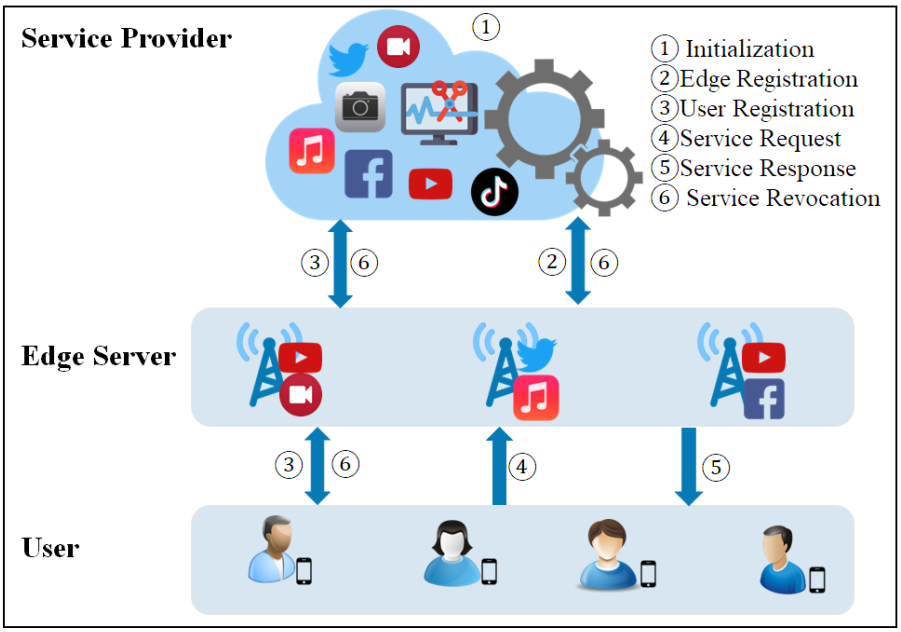


Figure 3.1: System Model

3.2 Models and Design Goals

In this section, we present the system and security models of distributed access control in MEC and the design goals of E-DAC.

3.2.1 System Model

The system model of dynamic MEC services is depicted in Fig. 3.1, which contains three types of parties: service providers, edge servers, and users.

- **Service Provider**: A service provider operates multiple cloud servers to offer a range of computing and storage services to its users. These services can be classified as either static or dynamic. Dynamic services require input data from users, such as videos, audio, or images, while static services, like content provision, do not require user input. In our framework, we assume that the service provider primarily offers dynamic services to registered users. The service provider has the capability to

authorize edge servers to deliver its services at the network edge exclusively to registered users. User registration is managed by the service provider, ensuring that only registered users can access the services on the edge servers. Moreover, the service provider has the authority to revoke service access of an edge server or a registered user if any misbehavior is detected.

- **Edge Server**: The edge server plays a crucial role by delivering the delegated services to registered users at the network edge. These edge servers can be the pre-deployed edge computing infrastructure, such as base stations and gateways.
- **User**: A user can register for the desired services with the end-devices through the service provider and subsequently gain access to the services on nearby edge servers or remote servers.

The service provider manages the registration and revocation of edge servers and users. After registration, the users and edge servers achieve 0-RTT mutual authentication with each other before the users accessing the services on the edge servers. According to these procedures, the distributed access control in MEC consists of six phases, namely, initialization, edge registration, user registration, service request, service response, and service revocation.

Initialization (INit): The service provider executes this algorithm to bootstrap the whole system to provide a set of services S. With the security parameter λ, the algorithm outputs the system parameter $\mathcal{P}$. The service provider chooses the master secret key msk and computes the public key pk for service initialization (Step ①).

Edge Registration (EReg): The edge server with an identity ID_e registers a set of services $S_e \subseteq S$ from the service provider. With the input of $(\mathcal{P}, msk, pk, ID_e, S_e)$, the service provider generates an edge service credential $Cred_s$ and returns the credential to the edge server for service authorization (Step ②).

User Registration (UReg): The user with an identity ID_u subscribes a set of services $S_u \subseteq S$ from the service provider. With the input of $(\mathcal{P}, msk, pk, ID_u, S_u)$, the service provider generates a service access credential $Cred_u$, and returns it to the user for service authorization (Step ③).

Service Request (SReq): To request a service $i \subseteq S_u$, with the input of $(\mathcal{P}, pk, i, m, ID_u, Cred_u)$, where m is the service data, the user generates a service request Req, containing the ciphertext of m, C, and sends the request to the network edge nodes (Step ④).

Service Response (SRes): With the input of $(\mathcal{P}, pk, Cred_s, Req)$, an edge server who receives the service request Req enforces implicit mutual authentication by using $Cred_s$ to

decrypt C in Req. If the authentication (decryption) fails, the algorithm outputs failure; otherwise, the algorithm outputs m, thus the edge service is allowed to access the service data to respond the service (Step ⑤).

Service Revocation (SRev): If an authorized edge server ID_e is compromised, the service provider revokes its service privilege with the input of $(\mathcal{P}, pk, ID_e, Cred_s)$; If an authorized user ID_u is compromised, the service provider revokes its service privilege with the input of $(\mathcal{P}, pk, ID_u, Cred_u)$.

The distributed access control in MEC must possess *Correctness*, which means that 1) an honest registered user can produce a valid service request for a service with her service access credential to authenticate to an honest edge server if it has registered that service, and 2) an honest registered edge server can use its valid edge service credential to authenticate to and provide services for the user, if it has registered that service. Therefore, *Correctness* of distributed access control in MEC is maintained under the conditions that i) both user and edge server have registered the service, and ii) the credentials of both user and edge server are valid, that is, the credentials are unexpired, unrevoked, and correctly issued by the service provider.

3.2.2 Security Model

The service provider provides its services, so it prefers to maintain good reputation and attract more users, rather than deviating from the committed service policies in the terms of services. So we assume that the service provider honestly serves its customers (including edge servers and users) for monetary benefits. Also, the basic security countermeasures, including firewalls, intrusion prevention systems, and virus detection systems, are deployed on the cloud servers to prevent outsider attackers from compromising them. However, it is hard to guarantee that all users or edge servers are completely honest. They may have the following misbehavior to disrupt the services.

- **Threats from Users**: We assume that users are not fully trusted in the purpose of accessing services without valid access credentials. (a) Unauthorized users may manipulate service access credentials to generate legitimate service requests or replay requests made by authorized users. These actions are categorized as either forgery attacks or replay attacks. (b) Authorized users might exploit expired service access credentials to gain access to services on the edge servers. (c) Revoked users can still access services on the edge servers with their revoked service access credentials. In summary, a malicious user engages in these behaviors with the intention of obtaining

a valid credential to successfully authenticate themselves with the edge server. This unauthorized access enables them to exploit resources on the edge servers for their purposes, leading to potential misuse and abuse of the system.

- **Threats from Edge Servers**: We assume that edge servers are not fully trusted in the purpose of providing services without valid credentials and accessing service data of users. (a) An unauthorized edge server has the ability to counterfeit an edge service credential, enabling it to gain access to sensitive service data belonging to users. (b) Even after being revoked, an edge server can continue to provide services to users by utilizing an outdated edge service credential. (c) A revoked edge server can collaborate with other revoked edge servers to illicitly acquire sensitive input data from users. In summary, a malicious or revoked edge server engages in these behaviors with the intent of masquerading as a delegated edge server, accessing users' sensitive data, and potentially compromising their data confidentiality. This can lead to fraudulent activities by unauthorized parties, causing harm to the honest users.

The adversaries, including malicious users or edge servers, aim to obtain valid credentials (edge service credentials or service access credentials) to access or provide services. To achieve these goals, they may learn knowledge from others' credentials, expired credentials, and revoked credentials. Therefore, to model attack capabilities of adversaries, we enable an adversary to adaptively query an edge service credential or an service access credential on behalf of a malicious edge server or a malicious user, respectively. According to the phases of distributed access control in MEC, an adversary is allowed to make the following queries:

- *EdgeReg Query*: The adversary adaptively queries $(\mathcal{P}, pk, ID_e, S_e)$ to the *EdgeReg* oracle, and the oracle runs the Edge Registration algorithm to generate and return an edge service credential $Cred_s$.
- *UserReg Query*: The adversary adaptively queries $(\mathcal{P}, pk, ID_u, S_u)$ to the *UserReg* oracle, and the oracle runs the User Registration algorithm to generate and return a service access credential $Cred_u$.
- *ServReq Query*: The adversary adaptively queries $(\mathcal{P}, pk, i, m, ID_u, Cred_u)$ with the service index $i \in S_u$ to the *ServReq* oracle, and the oracle runs the Service Request algorithm to generate and return a service request Req.
- *UserRev Query*: The adversary adaptively queries $(\mathcal{P}, pk, ID_u, Cred_u)$ to the *UserRev* oracle, and the oracle adds it to the *userRevocationTable*.

- *ServRev Query*: The adversary adaptively queries $(\mathcal{P}, pk, ID_e, Cred_s)$ to the *ServRev* oracle, and the oracle revokes the service privilege.
- *AuthVer Query*: The adversary adaptively queries the service request Req and the edge service credential $Cred_s$ to the *AuthVer* oracle, and the oracle returns failure or m to show whether Req can succeed the verification with $Cred_s$, i.e., $Cred_s$ can decrypt the ciphertext C.

The goal of an adversary is to succeed the authentication of the edge servers impersonating a user to access the requested service or to succeed the authentication of the user impersonating an edge server to access the service data of the user. To safeguard MEC against potential adversaries, the access control of users and edge servers should be guaranteed.

- **User Access Control**: 1) Authorization: Prior to accessing the desired services on the edge server, a user must be authorized by the service provider using a service access credential. 2) Authentication: The edge server plays a crucial role in efficiently authenticating the user, ensuring that it possesses the necessary authorization through the service access credential. This step is essential to grant access to the services. 3) Revocation: To prevent resource consumption by malicious or inactive users, it is imperative to promptly revoke a user's service access authorization. This ensures that only authorized individuals can continue to utilize the resources provided by the service provider or edge servers.
- **Edge Server Access Control**: 1) Authorization: Before providing services to users, an edge server must obtain authorization from the service provider through an edge service credential. This step ensures that only authorized edge servers can offer services. 2) Authentication: The user plays a crucial role in efficiently authenticating the edge server to verify its authorization to provide services and access the user's input data. This authentication process ensures that users interact only with trusted and authorized edge servers. 3) Revocation: It is essential to promptly revoke the service authorization of an edge server to prevent malicious servers from accessing users' input data. This proactive revocation measure safeguards the privacy and security of users' sensitive information.

Authorization means that there is no adversary to produce valid credentials for a specific user or edge server, although it may acquire others' credentials or have expired or revoked credentials. To secure authorization, it is impossible for the adversary to forge a valid

credential to serve as an edge server or impersonate a user. The unforgeability property should be guaranteed for the service access credentials and edge service credentials.

Unforgeability of Service Access Credentials: The advantage in forging a service access credential of a forger algorithm $\mathcal{A}$, given access to a set of oracles $\mathbb{S}^u$ = *UserReg, ServReq, UserRev, AuthVer* oracles is given as:

$$\mathbf{Adv}Auth^u_{\mathcal{A}} \stackrel{def}{=} \mathbf{Pr}\left[\begin{array}{c} SRes(\mathcal{P}, pk, Cred_s, Req) = m : \\ (\mathcal{P}, msk, pk) \leftarrow INit(\lambda); \\ Cred_s \leftarrow EReg(\mathcal{P}, msk, pk, ID_e, S_e); \\ S_u \cap S_e \neq \emptyset; \\ Cred_u \leftarrow \mathcal{A}^{\mathbb{S}^u}(\mathcal{P}, pk, ID_u, S_u); \\ i \in S_u \cap S_e; \\ Req \leftarrow \mathcal{A}^{\mathbb{S}^u}(\mathcal{P}, pk, i, m, ID_u, Cred_u) \end{array}\right].$$

Here, the adversary $\mathcal{A}$ is allowed to query the oracles adaptively: any of its queries may depend on previous answers, but it cannot query a service access credential for the service i that is contained in S_e.

Definition 1. The service access credential is unforgable in distributed access control in MEC if the advantage to forge a service access credential for an adversary, i.e., $\mathbf{Adv}Auth^u_{\mathcal{A}}$, is negligible.

Unforgeability of Edge Service Credentials: The advantage in forging an edge service credential of a forger algorithm $\mathcal{A}$, given access to a set of oracles $\mathbb{S}^s$ = *UserReg, EdgeReg, EdgeRev, AuthVer* oracles is given as:

$$\mathbf{Adv}Auth^s_{\mathcal{A}} \stackrel{def}{=} \mathbf{Pr}\left[\begin{array}{c} SRes(\mathcal{P}, pk, Cred_s, Req) = m : \\ (\mathcal{P}, msk, pk) \leftarrow INit(\lambda); \\ Cred_u \leftarrow UReg(\mathcal{P}, msk, pk, ID_u, S_u); \\ i \in S_u \cap S_e; \\ Req \leftarrow SReq(\mathcal{P}, pk, i, m, ID_u, Cred_u); \\ Cred_s \leftarrow \mathcal{A}^{\mathbb{S}^s}(\mathcal{P}, pk, ID_e, S_e) \end{array}\right].$$

Here, the adversary $\mathcal{A}$ is allowed to query the oracles adaptively: any of its queries may depend on previous answers, but it cannot query an edge service credential for the service i that is requested by the user.

Definition 2. The edge service credential is unforgable in distributed access control in MEC if the advantage to forge an edge service credential for an adversary, i.e., $\mathbf{Adv}Auth^s_{\mathcal{A}}$, is negligible.

$$\mathbf{Adv}IND^{d}_{\mathcal{A}} \stackrel{def}{=} \mathbf{Pr}\left[b = b^{*} : \begin{array}{c} (\mathcal{P}, msk, pk) \leftarrow INit(\lambda); \\ Cred_u \leftarrow UReg(\mathcal{P}, msk, pk, ID_u, S_u); \\ m_0, m_1 \leftarrow \mathcal{A}(\mathcal{P}, pk); \\ b \leftarrow \{0,1\}; \\ i \in S_u; \\ Req \leftarrow SReq(\mathcal{P}, pk, i, m_b, Cred_u); \\ m_{b^*} \leftarrow \mathcal{A}^{\mathbb{S}^d}(\mathcal{P}, pk, Req) \end{array} \right] - \frac{1}{2}. \tag{3.1}$$

Mutual authentication is satisfied if both edge service credential and service access credential are valid. The verification of both credentials is achieved in the 0-RTT mutual authentication. During the authentication, the service data of the user is encrypted by the service access credential to produce the service request, which is decrypted by the edge service credential for mutual authentication. The success of correct decryption indicates that both credentials are valid, so the correctness of data encryption guarantees authentication correctness.

In addition, the confidentiality of service data should be supported for protecting the service data of users. To model the confidentiality of the service data, semantic security should be achieved.

Semantic Security of Service Data: The semantic security of the service data is the indistinguishability against chosen plaintext attacks. The advantage to distinguish two plaintexts for an adversary with the knowledge of the ciphertext in the service request and the queries of oracles $\mathbb{S}^d$ = *UserReg, EdgeReg, ServReq, UserRev, EdgeRev, AuthVer* oracles is given in Equ. 1.

With the initial setup of the semantic security model and the service access credential $Cred_u$ of the targeted user, the challenge of indistinguishability against chosen plaintext attacks is illustrated using the given two messages (m_0, m_1) and the output of SReq Req generated from m_b. The response of the successful adversary should identify that Req is computed from m_{b^*}, where $b = b^*$. The adversary is allowed to query all possible oracles in $\mathbb{S}^d$ that can improve attack capabilities adaptively: any of its queries may depend on previous answers, but it cannot query an edge service credential for the service i that is requested by the user. Finally, if the adversary has more than a 1/2 probability of succeeding in the game, i.e., correctly outputting $b^* = b$, the adversary has the advantage of corrupting the semantic security of the service data.

Definition 3. The service data is confidential in distributed access control in MEC if the advantage to output a b^* for an adversary, such that $b^* = b$, i.e., $\mathbf{Adv}IND^{d}_{\mathcal{A}}$, is

negligible.

3.2.3 Design Goals

We summarize that the desirable properties of E-DAC include:

- User Access Authorization and Revocation: Users are authorized to access specific services through service access credentials, and such authorization can be revoked by the service provider when necessary.
- Edge Server Access Control: Edge servers are authorized by the service provider to offer services using edge service credentials, and this delegated authorization can be revoked when required.
- Mutual Authentication: Mutual authentication between users and edge servers is supported through the use of service access credentials and edge service credentials, ensuring that both parties are authenticated before service delivery.
- Data Confidentiality: The confidentiality of service data is preserved to protect users from potential eavesdropping and unauthorized data disclosure by adversaries.

3.3 E-DAC

In this section, we present the overview of E-DAC and its detailed design.

3.3.1 Overview of E-DAC

The framework is derived from KAC [107], with the integration of revocable broadcast encryption [108]. KAC, proposed by Chu et al. [109], is designed for scalable data sharing in cloud storage with the distinguished feature of efficient and flexible data access delegation. In E-DAC, we extend it to support flexible service management and authorization of the service provider. Specifically, the decryption keys of the encrypted files in cloud data sharing are used as the access keys of the services for the edge server in E-DAC. Each service has an access key. With the access key of a service, the edge server can recover the service data sent by the registered user. The authorization key is produced by the service provider for the edge server by aggregating the access keys of the set of authorized services.

The authorization key of the edge server has a constant size because of the aggregation of access keys. In addition, to prevent the forgery of an authorization key by combining two valid authorization keys, the authorization key is generated with the public key of the edge server and the set of services delegated to the edge server, thus the combination of two authorization keys of different edge servers is not valid.

The secret sharing-based broadcast encryption [108] is utilized to achieve service access authorization for users. The service key of the service provider is shared between a registered user and an edge server, and both possess different shares of the service key. The successful recovery of the service key in authentication achieves mutual authentication between the user and the edge server. Here, the service key and the authorization key are used by the edge server to decrypt the service data of the user, so that an "always-online" cloud authentication server, i.e., the service provider, is not involved in authentication. Furthermore, to revoke the service authorization to a dropout or rogue edge server, the service provider can release its secret share of the service key. After that, the user uses the released secret share to generate the service request, and the revoked edge server cannot access the service data in the request. The other honest edge servers do not need to update their individual authorization keys, thus the revocation cost is low.

3.3.2 Detailed E-DAC

The detailed construction of E-DAC is shown below. The notations of E-DAC are presented in Table 3.1.

3.3.2.1 Initialization

The system is initialized by a service provider $\mathcal{SP}$, who takes the security level parameter 1^λ and the number of its offered services n as input in the steps below.

Parameter Setup. The n services provided by $\mathcal{SP}$ are denoted as the index set $S = \{1, \cdots, n\}$. According to the security level parameter 1^λ, $\mathcal{SP}$ chooses a large prime p, where $2^\lambda \leq p \leq 2^{\lambda+1}$, and generates two multiplicative cyclic groups G, G_T of the order p. g and g_0 are the generators chosen from the group G. There is a bilinear mapping $\hat{e} : G \times G \to G_T$ for G and G_T. Also, $\mathcal{SP}$ generates hash functions $H : \{0,1\}^* \to G_T$ and $H_1 : \{0,1\}^* \to Z_p^*$. Next, with the index set of services, $\mathcal{SP}$ randomly chooses $\alpha \in Z_p^*$ and calculates $g_i = g^{\alpha^i}$, where $i \in \{1, \cdots, n, n+2, \cdots, 2n\}$. Finally, $\mathcal{SP}$ outputs the system parameters $\mathcal{P} = \langle G, G_T, e, H, H_1, g, g_0, g_1, \cdots, g_n, g_{n+2}, \cdots, g_{2n} \rangle$ and erases α safely after parameter setup.

Table 3.1: Notations of E-DAC

Symbol	Description
n	Number of services provided by SP
$\mathcal{S}$	Set of all service indices, $\mathcal{S} = \{1, \ldots, n\}$
$\mathcal{S}_e$	Set of services delegated to edge server E
$\mathcal{S}_u$	Set of services subscribed to by user U
i	Index of a requested service
α	Random secret used to generate the service parameters
γ	Master secret key of the service provider
msk, pk	Master secret key and corresponding public key
s	Secret service key
shr_{sp}	Set of protected shares maintained by SP
ID_e, ID_u	Identities of an edge server and a user, respectively
ν, V	Private and public keys of an edge server, where $V = g_0^{\nu}$
π	Non-interactive zero-knowledge proof of knowledge of ν
$K_{\mathcal{S}_e}$	Aggregate authorization key for services in $\mathcal{S}_e$
$(x_e, f(x_e))$	Secret service key share assigned to edge server E
T_i	Expiration time of user authorization for service i
shr_{ui}	Protected user share for service i
shr_r	Set of randomized protected shares in a service request
τ	Random value used for service data encryption
T_s, T_c	Request timestamp and current system time, respectively
$userTable$	Table containing registered user credentials
$serverTable$	Table containing registered edge server credentials
$userRevocationTable$	Table containing revoked user identities

Key Generation. $\mathcal{SP}$ randomly chooses $\gamma \in Z_p^*$ as the master secret key $msk = \gamma$ and computes g^{γ} as the public key $pk = g^{\gamma}$. msk is used to generate the authorization keys for edge servers in registration. Also, $\mathcal{SP}$ selects $s \in Z_p^*$ as the service key and computes g^s. The service key is used to generate the secret shares for users in registration.

Polynomial Generation. $\mathcal{SP}$ generates a polynomial $f(x)$ of a degree t, $f(x) = \sum_{i=0}^{t} a_i x^i$, where the coefficient $a_0 = s$ and other coefficients $a_i \in Z_p^*$, for $i = 1, \cdots, t$ are random. t is the revocation threshold.

Service Provider Share Generation. According to the polynomial $f(x)$, $\mathcal{SP}$ computes $t-1$ secret shares $\langle (x_{sp_1}, f(x_{sp_1})), \cdots, (x_{sp_{t-1}}, f(x_{sp_{t-1}}))\rangle$, where x_{sp_i} are randomly selected from Z_p^*, for $i = 1, \cdots, t-1$, and $x_{sp_i} \neq x_{sp_j}$, when $i \neq j$. With $t-1$ secret shares,

$\mathcal{SP}$ computes $t-1$ protected shares $shr_{sp} = \langle (x_{sp_1}, g^{f(x_{sp_1})}), \cdots, (x_{sp_{t-1}}, g^{f(x_{sp_{t-1}})}) \rangle$.

3.3.2.2 Edge Registration

An edge server $\mathcal{E}$ can provide the services of the service provider $\mathcal{SP}$ to the connected users under the delegation of $\mathcal{SP}$, but it has to register the services on $\mathcal{SP}$ to obtain authorization in the following steps.

Registration Request. $\mathcal{E}$ selects a random value $\nu \in Z_p^*$ to generate $V = g_0^\nu$ as its public key and sends its identity information ID_e, the public key V, and the index set of the requested services $S_e \subseteq S$ to $\mathcal{SP}$ via a secure channel, along with a zero-knowledge proof of the private key ν expressed in Camenisch-Stalder notation [110]: $\pi \leftarrow NIZK\{\nu : V = g_0^\nu\}$.

Aggregate Key Generation. Upon receiving the registration request from $\mathcal{E}$, $\mathcal{SP}$ verifies if π is valid. If not, $\mathcal{SP}$ aborts and returns failure. Otherwise, $\mathcal{SP}$ checks whether V has been appeared. If yes, i.e., the edge server ID_e has been registered, $\mathcal{SP}$ aborts and returns failure. Otherwise, based on the index set S_e with the service indices js, $\mathcal{SP}$ calculates the access keys of the services in S_e as g_{n+1-j}^γ, for $j \in S_e$, and further generates an aggregate authorization key K_{S_e} for $\mathcal{E}$ as $K_{S_e} = (V \prod_{j \in S_e} g_{n+1-j})^\gamma$. K_{S_e} is the authorization key for $\mathcal{E}$ on the services in S_e and used to decrypt the encrypted input data of the users who access the services in S_e.

Edge Share Generation. $\mathcal{SP}$ chooses $x_e \in Z_p^*$ and computes $f(x_e)$ to obtain the edge share of the service key $(x_e, f(x_e))$ for $\mathcal{E}$, where $x_e \neq x_{sp_i}$, for $i = 1, \cdots, t-1$.

Edge Credential Generation. $\mathcal{SP}$ produces the edge service credential as $Cred_s = \langle ID_e, S_e, K_{S_e}, (x_e, f(x_e)) \rangle$ and returns it to $\mathcal{E}$ via a secure channel. $Cred_s$ is securely stored by $\mathcal{E}$. Meanwhile, $\mathcal{SP}$ maintains $\mathcal{E}$'s credential $Cred_s$ in a secure table $serverTable$.

3.3.2.3 User Registration

Before accessing some services $S_u \subseteq S$ offered by the service provider $\mathcal{SP}$, a user $\mathcal{U}$ needs to subscribe them, so the user registers these services on $\mathcal{SP}$ to acquire authorization.

Registration Request. To subscribe services, $\mathcal{U}$ sends its identity information ID_u and the set of the indices of the interested services $S_u \subseteq S$ to $\mathcal{SP}$ via a secure channel.

User Share Generation. For the set of service indices is, $\mathcal{SP}$ determines the expiration time of the service i in S_u, and computes $x_{u_i} = H_1(ID_u||i||T_i)$, where ID_u is the identity information of $\mathcal{U}$ and T_i is the expiration time of the service i in S_u. $\mathcal{SP}$ further

computes the user share of each registered service $(x_{u_i}, f(x_{u_i}))$ and the protected shares $shr_{u_i} = (x_{u_i}, g^{f(x_{u_i})})$ for each service $i \in S_u$. $\mathcal{SP}$ sets $shr_u = \langle shr_{u_i} \rangle$ and $T = \langle T_i \rangle$ for $i \in S_u$.

User Credential Generation. $\mathcal{SP}$ generates the service access credential $Cred_u = \langle ID_u, S_u, g^s, shr_{sp}, shr_u, T \rangle$ from the user shares and the service provider returns it to $\mathcal{U}$ via a secure channel. Upon receiving $Cred_u$, $\mathcal{U}$ securely maintains $Cred_u$. Meanwhile, $\mathcal{SP}$ stores $Cred_u$ in a secure table $userTable$.

3.3.2.4 Service Request

The registered user $\mathcal{U}$ can request a subscribed service $i \in S_u$ from the service provider $\mathcal{SP}$ or an edge server $\mathcal{E}$ if it has been delegated to provide such service in the following steps:

Randomization. $\mathcal{U}$ selects a random value $k \in Z_p^*$ to calculate g^k and randomizes the protected shares in $Cred_u$ as $shr_r = \langle (x_{u_i}, g^{kf(x_{u_i})}), (x_{sp_1}, g^{kf(x_{sp_1})}), \cdots, (x_{sp_{t-1}}, g^{kf(x_{sp_{t-1}})}) \rangle$.

Data Encryption. The input data $m \in G_T$ is the service data that $\mathcal{U}$ submits to $\mathcal{SP}$ or $\mathcal{E}$ for accessing the requesting service i. To achieve 0-RTT mutual authentication, $\mathcal{U}$ encrypts the input data m of the service i by randomly selecting $\tau \in Z_p^*$ and computing the ciphertext of m as $C = \langle g^\tau, g_0^\tau, (pk \cdot g_i)^\tau, (m||H(m)) \cdot \hat{e}(g_1, g_n)^\tau \cdot \hat{e}(g^k, g^s)^\tau \rangle$.

Request Broadcast. $\mathcal{U}$ sets its service request $Req = \langle ID_u, i, shr_r, g^k, C, T_i, T_s \rangle$, where i is the service index, T_i is the expiration time of the service i, and T_s is a timestamp, and broadcasts the request Req into the edge network.

3.3.2.5 Service Response

After receiving a service request Req from a user $\mathcal{U}$, the edge server $\mathcal{E}$ executes the following way to respond the request.

Request Verification. From the service request Req, $\mathcal{E}$ extracts the index of the requested service i and checks if the requested service is in the scope of its capability. If it is not authorized to provide this service, $\mathcal{E}$ forwards Req to other edge servers; otherwise, it handles the request in the following steps: $\mathcal{E}$ first compares the timestamp T_s with the current time T_c to ensure the request freshness. Then, $\mathcal{E}$ extracts $(x_{u_i}, g^{kf(x_{u_i})})$ from shr_r and verifies whether $H_1(ID_u||i||T_i) = x_{u_{id}}$ to check user identity, service, and expiration time, respectively. If holds, $\mathcal{E}$ searches ID_u in its $userRevocationTable$ to ensure $\mathcal{U}$ has not been revoked and compares T_i with T_c to ensure that the service authorization is

not expired. If any check fails, $\mathcal{E}$ returns failure and aborts. Otherwise, $\mathcal{E}$ executes data decryption.

Data Decryption. To decrypt the input data m, $\mathcal{E}$ recovers the protected and randomized version of the service key g^{ks} from its share and the shares in the service request Req as follows: $\mathcal{E}$ first utilizes its secret share of the service key $(x_e, f(x_e))$ and g^k to compute the $(t+1)$-th protected and randomized share $(x_e, (g^k)^{f(x_e)})$. With the t protected and randomized shares shr_r in Req, $\mathcal{E}$ recovers g^{ks} through polynomial interpolation:

$$g^{ks} = g^{ka_0} = g^{k\sum_{i=0}^{t} f(x_i)\lambda_i} = \prod_{i=0}^{t} (g^{kf(x_i)})^{\lambda_i},$$

where $\lambda_i = \prod_{j=0, j\neq i}^{t} \frac{-x_i}{x_j - x_i}$. Here $x_0 = x_{u_i}$, $x_1 = x_{sp_1}$, $\cdots$, $x_{t-1} = x_{sp_{t-1}}$, $x_t = x_e$; $f(x_0) = f(x_{u_i})$, $f(x_1) = f(x_{sp_1})$, $\cdots$, $f(x_{t-1}) = f(x_{sp_{t-1}})$, $f(x_t) = f(x_e)$. After recovering g^{ks}, $\mathcal{E}$ decrypts $C = \langle c_1, c_2, c_3, c_4 \rangle$ to obtain $m||H(m)$ by using K_{S_e} and g^{ks} as follows:

$$m||H(m) = c_4 \cdot \frac{\hat{e}(K_{S_e} \cdot \prod_{j\in S_e, j\neq i} g_{n+1-j+i}, c_1)}{\hat{e}(pk, c_2)^{\nu} \hat{e}(\prod_{j\in S_e} g_{n+1-j}, c_3) \cdot \hat{e}(g^{\tau}, g^{ks})}.$$

Finally, $\mathcal{E}$ verifies whether the recovered m is correct based on the hash value. If yes, $\mathcal{E}$ replies the request; otherwise, $\mathcal{E}$ aborts and returns failure.

Request Response. If the input data m is decrypted successfully, $\mathcal{E}$ fulfills u's requested service.

3.3.2.6 Service Revocation

Service revocation enables the service provider $\mathcal{SP}$ to withdraw the access ability of the users or the ability of the edge servers to provide services if any misbehavior is identified.

User Revocation. If the user $\mathcal{U}$'s authorization needs to be revoked, the service provider $\mathcal{SP}$ checks the set of the edge servers E who serve $\mathcal{U}$ from $serverTable$. Then, $\mathcal{SP}$ sends a revocation notification and $\mathcal{U}$'s identity to each edge server in E and deletes $\mathcal{U}$'s user credential in $userTable$. Once receiving the revocation notification, the edge servers add $\mathcal{U}$'s identity in their $userRevocationTable$.

Edge Server Revocation. To revoke the edge server $\mathcal{E}$'s service authorization, $\mathcal{SP}$ finds $\mathcal{E}$'s credential $Cred_s$, extracts the edge share $(x_e, f(x_e))$, and replaces one of the shares shr_{sp} with $(x_e, g^{f(x_e)})$, which is generated from the edge share $(x_e, f(x_e))$. Then, $\mathcal{SP}$ sends the updated shares shr^*_{sp} to all registered users. Finally, $\mathcal{SP}$ deletes $\mathcal{E}$'s credential

$$
\begin{aligned}
& c_4 \cdot \frac{\hat{e}(K_{S_e} \cdot \prod_{j \in S_e, j \neq i} g_{n+1-j+i}, c_1)}{\hat{e}(pk, c_2)^{\nu} \hat{e}(\prod_{j \in S_e} g_{n+1-j}, c_3) \cdot \hat{e}(g^{\tau}, g^{ks})} \qquad (3.2) \\
= \; & (m||H(m)) \cdot \hat{e}(g_1, g_n)^{\tau} \cdot \hat{e}(g^k, g^s)^{\tau} \frac{\hat{e}(\prod_{j \in S_e} g_0^{\nu\gamma} g_{n+1-j}^{\gamma}, g^{\tau}) \hat{e}(\prod_{j \in S_e, j \neq i} g_{n+1-j+i}, g^{\tau})}{\hat{e}(g^{\gamma}, g_0)^{\tau\nu} \hat{e}(\prod_{j \in S_e} g_{n+1-j}, c_3) \cdot \hat{e}(g^{\tau}, g^{ks})} \\
= \; & (m||H(m)) \cdot \hat{e}(g_1, g_n)^{\tau} \frac{\hat{e}(\prod_{j \in S_e} g_{n+1-j}^{\gamma}, g^{\tau}) \hat{e}(\prod_{j \in S_e, j \neq i} g_{n+1-j+i}, g^{\tau})}{\hat{e}(\prod_{j \in S_e} g_{n+1-j}, (pk \cdot g_i)^{\tau})} \\
= \; & (m||H(m)) \cdot \hat{e}(g_1, g_n)^{\tau} \frac{\hat{e}(\prod_{j \in S_e} g_{n+1-j}^{\gamma}, g^{\tau}) \hat{e}(\prod_{j \in S_e, j \neq i} g_{n+1-j+i}, g^{\tau})}{\hat{e}(\prod_{j \in S_e} g_{n+1-j}, g^{\gamma\tau}) \hat{e}(\prod_{j \in S_e} g_{n+1-j}, g_i^{\tau})} \\
= \; & (m||H(m)) \cdot \hat{e}(g_1, g_n)^{\tau} \frac{\hat{e}(\prod_{j \in S_e, j \neq i} g_{n+1-j+i}, g^{\tau})}{\hat{e}(\prod_{j \in S_e} g_{n+1-j+i}, g^{\tau})} \\
= \; & (m||H(m)) \cdot \frac{\hat{e}(g_1, g_n)^{\tau}}{\hat{e}(g_{n+1}, g^{\tau})} \\
= \; & (m||H(m)) \cdot \frac{\hat{e}(g^{\alpha}, g^{\alpha^n})^{\tau}}{\hat{e}(g^{\alpha^{n+1}}, g^{\tau})} \\
= \; & m||H(m).
\end{aligned}
$$

in $serverTable$. Because the number of the shares shr_{sp} is $t-1$, the maximal number of edge servers that can be revoked is $t-1$. If the number of revoked edge servers reaches $t-1$, $\mathcal{SP}$ should update the service key and re-distribute shares of the service key to users and edge servers.

Correctness: The correctness of the proposed E-DAC is demonstrated in Equ. 3.2. The correctness ensures correct mutual authentication between the user and the edge server.

3.4 Security Analysis

In this section, we analyze the unforgeability of service access credentials, the unforgeability of edge service credentials, and the confidentiality of the service data of users. The mutual authentication is achieved if the correctness of E-DAC holds, which is built upon the fact that the service access credential of the user and the edge service credential of the edge server are correctly produced. Here, we prove the unforgeabilty of both credentials of the

user and the edge server.

Theorem 1 (*Unforgeability of Service Access Credentials*). Assume the CDH problem is hard, the service access credential in E-DAC is unforgeable against the chosen service attacks.

Proof. The unforgeability of service access credentials can be reduced to the CDH assumption, that is, within non-negligible advantage, if there is no probabilistic polynomial-time algorithm to solve CDH problem, then, the service access credential in E-DAC is unforgeable against the chosen service attacks. Suppose that an adversary $\mathcal{A}$ can break the unforgeability of service access credentials with a non-negligible probability, then we can construct an algorithm $\mathcal{B}$ to solve the CDH problem: given $g, g^a, g^b \in G$, where $a, b \in Z_p$, to compute $g^{ab} \in G$. $\mathcal{B}$ simulates a challenger to interact with $\mathcal{A}$ in the following way:

In initialization, $\mathcal{B}$ setups the parameters $(\mathcal{P}, S, V, \gamma, g^\gamma)$ and calculates $g_0 = g^\beta$, $g_i = g^{\alpha^i}$, where $\alpha, \beta \in Z_p^*$ and $i \in \{1, \cdots, n, n+2, \cdots, 2n\}$. The service key g^s is set as g^a, and the points $(x_{sp_1}, y_{sp_1}), \cdots, (x_{sp_{t-1}}, y_{sp_{t-1}})$ are randomly chosen. $\mathcal{B}$ also selects $S_e \in S$ to compute $K_{S_e} = (V \prod_{j \in S_e} g_{n+1-j})^\gamma$ and randomly chooses the point (x_e, y_e). Finally, $\mathcal{B}$ sends $(\mathcal{P}, S, g^\gamma, g^a, g^\beta, g^\alpha, \cdots, g^{\alpha^n}, g^{\alpha^{n+2}}, \cdots, g^{\alpha^{2n}})$ to $\mathcal{A}$.

$\mathcal{B}$ programs oracles to answer the queries from $\mathcal{A}$. To ensure the consistency, it maintains a list of tuples to keep the queries and corresponding responses. The queries are answered as follows:

- When receiving the $UserReg$ query $(\mathcal{P}, g^\gamma, ID_u, S_u)$, $\mathcal{B}$ randomly selects (x_{u_i}, y_{u_i}), computes $(x_{u_i}, g^{y_{u_i}})$, $(x_{sp_1}, g^{y_{sp_1}})$, $\cdots$, $(x_{sp_{t-1}}, g^{y_{sp_{t-1}}})$, and returns them to $\mathcal{A}$, along with g^a and a random T, that is, $Cred_u$.

- When receiving the $ServReq$ query $(\mathcal{P}, g^\gamma, i, m, ID_u, Cred_u)$, where $i \in S_u$, $\mathcal{B}$ randomizes $Cred_u$ to produce $shr_r = \langle (x_{u_i}, (g^b)^{y_{u_i}}), (x_{sp_1}, (g^b)^{y_{sp_1}}), \cdots, (x_{sp_{t-1}}, (g^b)^{y_{sp_{t-1}}}) \rangle$, uses (x_{u_i}, y_{u_i}), (x_e, y_e), $(x_{sp_1}, y_{sp_1}), \cdots, (x_{sp_{t-1}}, y_{sp_{t-1}})$ to reconstruct the polynomial $f(x) = s' + \sum_{i=1}^{t} a_i x^i$ and obtain s'. Then, $\mathcal{B}$ randomly chooses $\tau \in Z_p^*$ to compute the ciphtext $C = \langle g^\tau, g_0^\tau, (pk \cdot g_i)^\tau, (m||H(m)) \cdot e(g_1, g_n)^\tau \cdot e(g^b, g^{s'})^\tau \rangle$. Finally, $\mathcal{B}$ returns $Req = \langle ID_u, i, shr_r, g^b, C, T, T_s \rangle$ to $\mathcal{A}$.

- When receiving the $UserRev$ query $(\mathcal{P}, pk, ID_u, Cred_u)$, $\mathcal{B}$ adds it to $userRevocationTable$.

- When receiving the $AuthVer$ query Req, $\mathcal{B}$ uses $Cred_s = \langle ID_e, S_e, K_{S_e}, (x_e, y_e) \rangle$ that is generated in initialization to verify the validity of Req and returns failure or m to $\mathcal{A}$.

Eventually, $\mathcal{A}$ produces a service access credential and the corresponding service request $Req^* = \langle ID_{u^*}, i^*, shr_r^*, g^b, C^*, T_{i^*}, T_s^* \rangle$, where $shr_r^* = ((x_{u_i}^*, (g^b)^{y_{u_i}^*}), (x_{sp_1}, (g^b)^{y_{sp_1}}), \cdots, (x_{sp_{t-1}}, (g^b)^{y_{sp_{t-1}}})$. If the service i^* has been queried or $i^* \notin S_e$, $\mathcal{B}$ returns failure. Otherwise, $\mathcal{B}$ uses $Cred_s = \langle ID_u, S_e, K_{S_e}, (x_e, y_e) \rangle$ to verify whether Req^* is valid or not, and thereby determines whether $Cred_u^*$ is valid or not. If $Cred_u^*$ is invalid, $\mathcal{B}$ returns failure; otherwise, $\mathcal{B}$ reconstructs g^{ab} as:

$$g^{ab} = g^{b\sum_{i=0}^{t} f(x_i)\lambda_i} = \prod_{i=0}^{t} (g^{bf(x_i)})^{\lambda_i},$$

where $\lambda_i = \prod_{j=0, j\neq i}^{t} \frac{-x_i}{x_j - x_i}$. Here $x_0 = x_{u_i}^*$, $x_1 = x_{sp_1}$, $\cdots$, $x_{t-1} = x_{sp_{t-1}}$, $x_t = x_e$; $f(x_0) = y_{u_i}^*$, $f(x_1) = y_{sp_1}$, $\cdots$, $f(x_{t-1}) = y_{sp_{t-1}}$, $f(x_t) = y_e$.

Thus, the CDH problem has been solved. We prove that if there is an adversary forges the service access credential, the CDH problem can be solved. However, the advantage to solve the CDH problem is negligible, the probability to generate a service access credential in E-DAC for an adversary is negligible.

Theorem 2 (*Unforgeability of Edge Service Credentials*) Assume the CDH problem is hard, the edge service credential in E-DAC is unforgeable against the chosen service attacks.

Proof. The unforgeability of edge service credentials can be reduced to the CDH assumption, that is, within non-negligible advantage, if there is no probabilistic polynomial-time algorithm to solve CDH problem, then, the edge service credential in E-DAC is unforgeable against chosen service attacks. Suppose that an adversary $\mathcal{A}$ can break the unforgeability of edge service credentials with a non-negligible probability, then we can construct an algorithm $\mathcal{B}$ to solve the CDH problem: given $g, g^a, g^b \in G$, where $a, b \in Z_p$, to compute $g^{ab} \in G$. $\mathcal{B}$ simulates a challenger to interact with $\mathcal{A}$ in the following way:

In initialization, $\mathcal{B}$ setups the parameters $(\mathcal{P}, S, g^s)$ and the polynomial $f(x)$ of a degree t, where the coefficient $a_0 = s$ and other coefficients $a_i \in Z_p^*$, for $i = 1, \cdots, t$, are randomly generated. $(x_{sp_1}, f(x_{sp_1})), \cdots, (x_{sp_{t-1}}, f(x_{sp_{t-1}}))$ are randomly chosen based on $f(x)$. The public key is set as g^a. $\mathcal{B}$ generates $g_0 = g^\beta$, $g_i = g^{\alpha^i}$, $h = g^b$, $h_0 = h^\beta$, and $h_i = (g^b)^{\alpha^i}$, where $\alpha, \beta \in Z_p^*$ and $i \in \{1, \cdots, n, n+2, \cdots, 2n\}$.

$\mathcal{B}$ randomly selects $(x_{u_i}, f(x_{u_i}))$ and computes $(x_{u_i}, g^{f(x_{u_i})})$, $(x_{sp_1}, g^{f(x_{sp_1})})$, $\cdots$, $(x_{sp_{t-1}}, g^{f(x_{sp_{t-1}})})$, which is set as $Cred_u$, together with (ID_u, S_u, g^s, T). Then, $\mathcal{B}$ selects the challenging service i^* and a random value $k \in Z_p^*$, and calculates shr_r in Req as $shr_r = \langle (x_{u_{i^*}}, g^{kf(x_{u_{i^*}})}), (x_{sp_1}, g^{kf(x_{sp_1})}), \cdots, (x_{sp_{t-1}}, g^{kf(x_{sp_{t-1}})}) \rangle$. Then, $\mathcal{B}$ randomly chooses $\tau \in Z_p^*$

to compute $C = \langle h^{\tau}, h^{\tau\beta}, (pk \cdot h_{i^*})^{\tau}, (m||H(m)) \cdot \hat{e}(h_1, h_n)^{\tau} \cdot \hat{e}(g^k, g^s)^{\tau} \rangle$, and sets $Req^* = \langle ID_u, i^*, shr_r, g^k, C, T, T_s \rangle$. Finally, $\mathcal{B}$ sends $(\mathcal{P}, S, g^a, g^s, (x_{sp_1}, f(x_{sp_1})), \cdots, (x_{sp_{t-1}}, f(x_{sp_{t-1}})), \{g_i, h_i\}_{i \in \{1, \cdots, n, n+2, \cdots, 2n\}})$ to $\mathcal{A}$.

$\mathcal{B}$ programs oracles to answer the queries from $\mathcal{A}$. To ensure the consistency, it maintains a list of tuples to keep the queries and corresponding responses. The queries are answered as follows:

- When receiving the $EdgeReg$ query $(\mathcal{P}, g^a, g^{\beta v}, ID_e, i)$, where v is randomly chosen in Z_p^*, $\mathcal{B}$ generates a random coin $c_i \in \{0, 1\}$, where the service $i \in S_e$. If $c_i = 0$, $\mathcal{B}$ computes $l_i = (g^a)^{\alpha^i}$, for $i \in \{1, \cdots, n, n+2, \cdots, 2n\}$, and $K_{S_e} = g^{a\beta v} \prod_{j \in S_e} l_{n+1-j}$, randomly chooses $(x_e, f(x_e))$, and returns $Cred_s = \langle ID_e, i, c_i, K_{S_e}, (x_e, f(x_e)) \rangle$ to $\mathcal{A}$; otherwise, $\mathcal{B}$ aborts and returns failure.

- When receiving the $UserReg$ query $(\mathcal{P}, g^a, ID_u, S_u)$, $\mathcal{B}$ randomly selects $(x_{u_i}, f(x_{u_i}))$ and computes $(x_{u_i}, g^{f(x_{u_i})})$, $(x_{sp_1}, g^{f(x_{sp_1})})$, $\cdots$, $(x_{sp_{t-1}}, g^{f(x_{sp_{t-1}})})$, which is set as $Cred_u$, together with (ID_u, S_u, g^s, T). Then, $\mathcal{B}$ selects the service $i \neq i^*$ and a random value $k \in Z_p^*$, and calculates shr_r in Req as $shr_r = \langle (x_{u_i}, g^{kf(x_{u_i})}), (x_{sp_1}, g^{kf(x_{sp_1})}), \cdots, (x_{sp_{t-1}}, g^{kf(x_{sp_{t-1}})}) \rangle$. If $c_i = 0$, $\mathcal{B}$ randomly chooses $\tau \in Z_p^*$ to compute $C = \langle g^{\tau}, g^{\tau\beta}, (pk \cdot g_i)^{\tau}, (m||H(m)) \cdot \hat{e}(g_1, g_n)^{\tau} \cdot \hat{e}(g^k, g^s)^{\tau} \rangle$. If $c_i = 1$, $\mathcal{B}$ randomly chooses $\tau \in Z_p^*$ to compute $C = \langle h^{\tau}, h^{\tau\beta}, (pk \cdot h_i)^{\tau}, (m||H(m)) \cdot \hat{e}(h_1, h_n)^{\tau} \cdot \hat{e}(g^k, g^s)^{\tau} \rangle$. Finally, $\mathcal{B}$ sets $Req = \langle ID_u, i, shr_r, g^k, c_i, C, T, T_s \rangle$ and returns to $\mathcal{A}$.

- When receiving the $EdgeRev$ query $(\mathcal{P}, pk, ID_e, Cred_s)$, $\mathcal{B}$ sends the secret share of the edge server $(x_e, f(x_e))$ to all the users.

- When receiving the $AuthVer$ query $(Cred_s, Req)$, $\mathcal{B}$ checks whether $c_i = 0$ or $c_i = 1$. If $c_i = 0$, $\mathcal{B}$ verifies Req and $Cred_s$ by using Service Response algorithm, and returns failure or m to $\mathcal{A}$. If $c_i = 1$, $\mathcal{B}$ returns failure and aborts.

Eventually, $\mathcal{A}$ produces an edge service credential $Cred_s^* = \langle ID_e^*, S_e^*, c_{i^*}, K_{S_e^*}, (x_e^*, f(x_e^*)) \rangle$, where $i^* \in S_e^*$. $\mathcal{B}$ uses Req^* to verify whether $Cred_s^*$ is valid or not. If not, $\mathcal{B}$ returns failure and aborts; otherwise, $\mathcal{B}$ checks if $c_i = 1$. If not, $\mathcal{B}$ returns failure and aborts; otherwise, $\mathcal{B}$ computes $g^{ab} = (K_{S_e^*})^{(\beta v + \sum_{i^* \in S_e^*} \alpha^{n+1-i^*})^{-1}}$.

Thus, the CDH problem has been solved. We prove that if there is an adversary forges the edge service credential, the CDH problem can be solved. However, the advantage to solve the CCDH problem is negligible, the probability to generate an edge service credential in E-DAC for an adversary is negligible.

Theorem 3 (*Semantic Security of Service Data*) Assume the d-BDHE and DBDH problems are hard, the service data in E-DAC is semantically secure.

The confidentiality of the service data can be reduced to the d-BDHE and DBDH assumptions, that is, within non-negligible advantage, if there is no probabilistic polynomial-time algorithm to solve both d-BDHE and DBDH problems, then, the service data in E-DAC is semantically secure. To prove this statement, we divide the theorem 3 into two lemmas.

Lemma 1. Assume that the DBDH problem is hard, $m' = (m||H(m))\hat{e}(g_1, g_n)^{\tau}$ in the ciphtext $C' = \langle g^{\tau}, g_0^{\tau}, (pk \cdot g_i)^{\tau}, m' \cdot \hat{e}(g^k, g^s)^{\tau} \rangle$ is indistinguishable against chosen plaintext attacks.

Proof. Suppose that an adversary $\mathcal{A}$ can break the semantic security with a non-negligible probability, then we can construct an algorithm $\mathcal{B}$ to solve the DBDH problem: given $g, g^a, g^b, g^c \in G$ and a random element $R \in G_T$, to decide whether $\hat{e}(g, g)^{abc} = R$. $\mathcal{B}$ simulates a challenger to interact with $\mathcal{A}$ in the following way:

In initialization, $\mathcal{B}$ setups the parameters $(\mathcal{P}, S, V, \gamma, g^{\gamma})$ and calculates $g_0 = g^{\beta}$, $g_i = g^{\alpha^i}$, where $\alpha, \beta \in Z_p^*$ and $i \in \{1, \cdots, n, n+2, \cdots, 2n\}$. The service key g^s is set as g^a, and the points $(x_{sp_1}, y_{sp_1}), \cdots, (x_{sp_{t-1}}, y_{sp_{t-1}})$ are randomly chosen. $\mathcal{B}$ also selects $S_e \in S$ to compute $K_{S_e} = (V \prod_{j \in S_e} g_{n+1-j})^{\gamma}$ and randomly chooses the point (x_e, y_e). Finally, $\mathcal{B}$ sends $(\mathcal{P}, S, g^{\gamma}, g^a, g^{\beta}, g^{\alpha}, \cdots, g^{\alpha^n}, g^{\alpha^{n+2}}, \cdots, g^{\alpha^{2n}})$ to $\mathcal{A}$.

$\mathcal{B}$ programs oracles to answer the queries from $\mathcal{A}$. To ensure the consistency, it maintains a list of tuples to keep the queries and corresponding responses. The queries are answered as follows:

- When receiving the $UserReg$ query $(\mathcal{P}, g^{\gamma}, ID_u, S_u)$, $\mathcal{B}$ randomly selects (x_{u_i}, y_{u_i}), computes $(x_{u_i}, g^{y_{u_i}})$, $(x_{sp_1}, g^{y_{sp_1}})$, $\cdots$, $(x_{sp_{t-1}}, g^{y_{sp_{t-1}}})$, and returns them to $\mathcal{A}$, along with g^a and a random T, that is, $Cred_u$.

- When receiving the $ServReq$ query $(\mathcal{P}, g^{\gamma}, m', Cred_u, i)$, where $i \in S_u$, $\mathcal{B}$ randomizes $Cred_u$ to produce $shr_r = \langle (x_{u_i}, (g^b)^{y_{u_i}}), (x_{sp_1}, (g^b)^{y_{sp_1}}), \cdots, (x_{sp_{t-1}}, (g^b)^{y_{sp_{t-1}}}) \rangle$, uses (x_{u_i}, y_{u_i}), (x_e, y_e), $(x_{sp_1}, y_{sp_1}), \cdots, (x_{sp_{t-1}}, y_{sp_{t-1}})$ to reconstruct the polynomial $f(x) = a' + \sum_{i=1}^{t} a_i x^i$ and obtain a'. Then, $\mathcal{B}$ computes the ciphtext $C' = \langle g^{\tau}, g^{\tau\beta}, (g^{\tau})^{\gamma + \alpha^i}, m' \cdot \hat{e}(g^b, g^{\tau})^{a'} \rangle$, where $\tau \in Z_p^*$. Finally, $\mathcal{B}$ returns $Req = \langle ID_u, i, shr_r, g^b, C', T, T_s \rangle$ to $\mathcal{A}$.

- When receiving the $UserRev$ query $(\mathcal{P}, pk, ID_u, Cred_u)$, $\mathcal{B}$ adds it to the $userRevocationTable$.

- When receiving the $AuthVer$ query Req, $\mathcal{B}$ uses $Cred_s = \langle S_e, K_{S_e}, (x_e, y_e)\rangle$ that is generated in initialization to verify the validity of Req and returns failure or m to $\mathcal{A}$.

For m_0 and m_1 chosen by $\mathcal{A}$, $\mathcal{B}$ randomly chooses $w \in \{0,1\}$ and encrypts m_w as $C_w = \langle g^c, g^{c\beta}, (g^c)^{\gamma+\alpha^i}, m'_w \cdot D\rangle$. $\mathcal{B}$ randomly generates $(x_{u_i}, (g^b)^{y_{u_i}})$, $(x_{sp_1}, (g^b)^{y_{sp_1}})$, $\cdots$, $(x_{sp_{t-1}}, (g^b)^{y_{sp_{t-1}}})\rangle$ to generate shr_r. (C_w, shr_r) are returned to $\mathcal{A}$, and $\mathcal{A}$ outputs w^*. It is obvious that if $\mathcal{A}$ can correctly tell w^*, such that $w^* = w$, $\mathcal{B}$ can distinguish whether $D = \hat{e}(g,g)^{abc}$ or $D = R$, so as to solve the DBDH problem.

Lemma 2. Assume that the d-BDHE problem is hard, the ciphtext $C = \langle g^\tau, g_0^\tau, (pk \cdot g_i)^\tau, (m||H(m)) \cdot \hat{e}(g_1, g_n)^\tau\rangle$ is indistinguishable against chosen plaintext attacks.

Proof. Suppose that an adversary $\mathcal{A}$ can break the semantic security with a non-negligible probability, then we can construct an algorithm $\mathcal{B}$ to solve the decisional d-BDHE problem: given a vector of $2n$ elements $(g, g^\alpha, g^{\alpha^2}, \cdots, g^{\alpha^n}, g^{\alpha^{n+2}}, g^{\alpha^{2n}}) \in G$ and a random element $R \in G_T$, to decide whether $\hat{e}(g,g)^{\alpha^{n+1}} = R$. $\mathcal{B}$ simulates a challenger to interact with $\mathcal{A}$ in the following way:

In initialization, $\mathcal{B}$ setups the parameters $(\mathcal{P}, S, \gamma, g^\gamma, g^s)$ and the polynomial $f(x)$ of a degree t. $(x_{sp_1}, f(x_{sp_1})), \cdots, (x_{sp_{t-1}}, f(x_{sp_{t-1}}))$ are randomly chosen based on $f(x)$. $\mathcal{B}$ sets $g_0 = g^\beta$, $g_i = g^{\alpha^i}$, where $\beta \in Z_p^*$ and $i \in \{1, \cdots, n, n+2, \cdots, 2n\}$. $\mathcal{B}$ sends them to $\mathcal{A}$.

$\mathcal{B}$ programs oracles to answer the queries from $\mathcal{A}$. To ensure the consistency, it maintains a list of tuples to keep the queries and corresponding responses. Because $\mathcal{B}$ possesses all the secret values (γ, s), except α, which is not needed in the generation of edge service credentials and service access credentials, $\mathcal{B}$ can answer all the oracles without any differences with the corresponding algorithms in E-DAC.

For m_0 and m_1 chosen by $\mathcal{A}$, $\mathcal{B}$ randomly chooses $w \in \{0,1\}$ and encrypts m_w as $C_w = \langle g^\tau, g^{\tau\beta}, (pk \cdot g_i)^\tau, (m_w||H(m_w)) \cdot D\rangle$, which is returned to $\mathcal{A}$. It is obvious that if $\mathcal{A}$ can correctly tell w^*, such that $w^* = w$, $\mathcal{B}$ can distinguish whether $D = \hat{e}(g^{\alpha^{n+1}}, g^\tau)$ or $D = R$, so as to solve the d-BDHE problem.

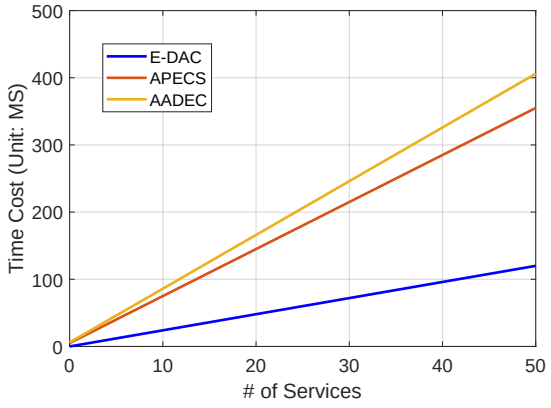
3.5 Performance Evaluation

In this section, we evaluate the computational and communication performance of E-DAC to demonstrate that our E-DAC outperforms the existing works.

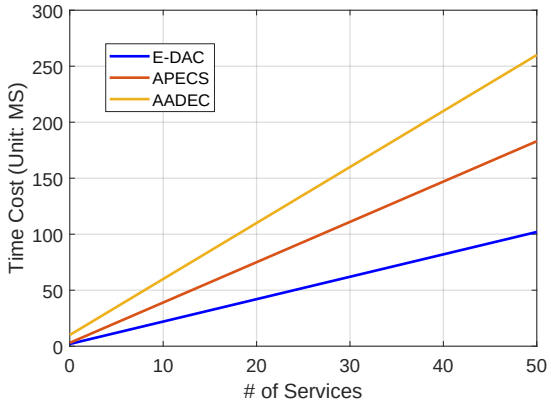
We show the computational overhead of E-DAC by quantifying the number of specific operations performed in G and G_T. The operations under evaluation include exponentiation in G, exponentiation in G_T, bilinear pairing operations, polynomial evaluation, and

Table 3.2: Performance of E-DAC

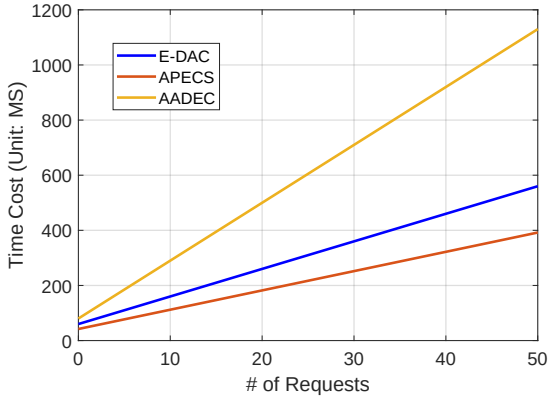
Phases	Entities	Computational Overhead	Communication Cost (Bits)	Execution Cost (ms)
Initialization	$\mathcal{SP}$	$(2n + t{+}1)E_G{+}(t\text{-}1)Ploy$	–	698
Edge Registration	$\mathcal{SP}$	$(v + 1)E_G{+}vMul_G{+}Ploy$	832	315
	Edge E_G	2	1346	
User Registration	$\mathcal{SP}$	$wE_G + wPloy$	$384 + 512t$	287
	User	–	64	
Service Request	User	$(t + 4)E_G + 2BP + E_{G_T} + Mul_G$	$2824 + 512t$	1153
Service Response	Edge	$(t + 2)E_G + 4BP + (t + 2v - 2)Mul_G$	–	
User Revocation	$\mathcal{SP}$	–	64	76
Edge Revocation	$\mathcal{SP}$	E_G	512	107



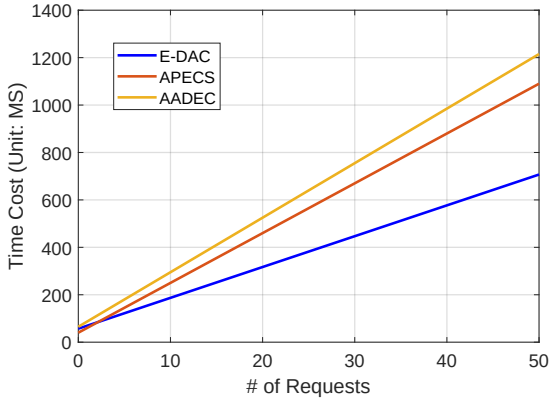
(a) Time Cost in Edge Server Registration



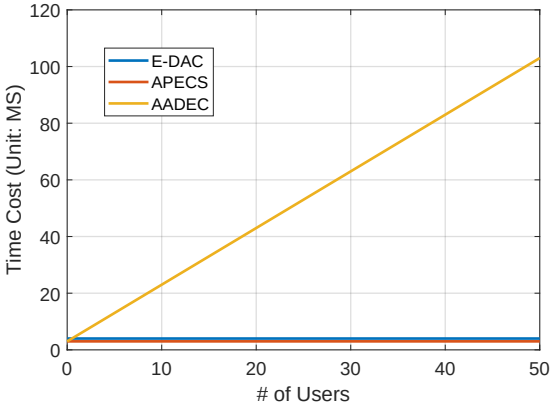
(b) Time Cost in User Registration



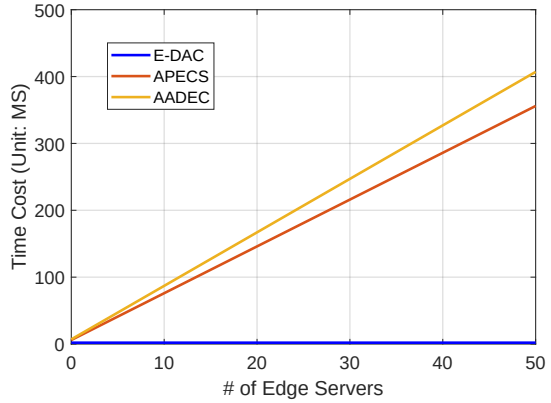
(c) Time Cost in Service Request



(d) Time Cost in Service Response



(e) Time Cost in User Revocation



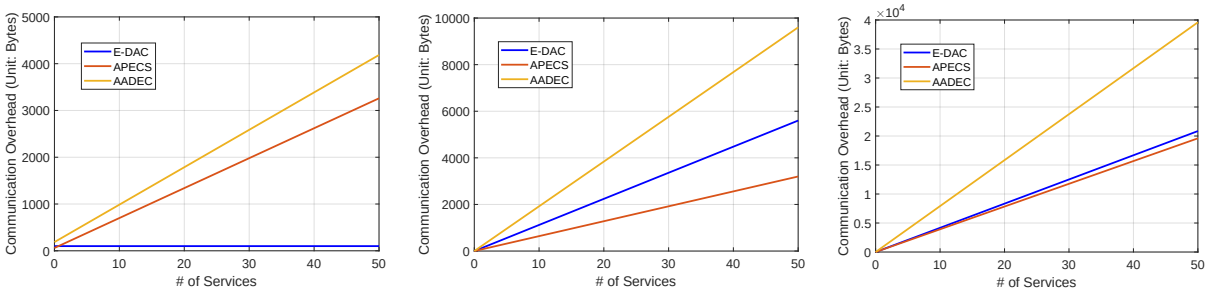
(f) Time Cost in Edge Server Revocation

Figure 3.2: Computational Overhead Comparison of E-DAC, APECS, and AADEC

multiplication in G. Notably, other operations like the hash operation and multiplication in Z_p^* are relatively less time-consuming in comparison. For convenience, we use E_G, E_{G_T}, BP, $Ploy$, and Mul_G to represent exponentiation in G, exponentiation in G_T, bilinear pairing, polynomial evaluation, and multiplication in G, respectively. The third column of Table 3.2 displays the number of computations involved in each phase of E-DAC. Let $v = |S_e|$ be the number of registered services of the edge server, and $w = |S_u|$ be the number of registered services of the user. Notably, the revocation process demonstrates high efficiency, requiring no time-consuming operations for either the edge server or the user. Moreover, in the service request phase, the bilinear pairing can be pre-computed, allowing the user to calculate $\hat{e}(g_1, g_n)$ and $\hat{e}(g, g^s)$ in advance, thereby expediting the service request generation.

We conducted comprehensive real-world experiments to measure the time consumption of the service provider, the edge server, and the user. The platform comprises a Precision 7920 Tower Workstation and two ThinkPad X1 Carbon laptops. The Precision 7920 Tower Workstation is configured with an Intel Xeon Gold 6242R 4.1GHz 20C CPU and 128GB RAM, and the ThinkPad X1 Carbon laptop is equipped with an Intel Core i7-1365U vPro 3.90 GHz CPU and 32GB RAM. The operations of the service provider are carried out on the Precision 7920 Tower Workstation, and the operations of the edge server and the user are conducted on two ThinkPad X1 Carbon laptops, respectively. The operating systems of these devices are Windows 11, and they are connected to a WiFi router to build a network. To implement number-theoretic based methods of cryptography, we utilized the JPBC library. For achieving bilinear pairing, we employed Type-A pairing. The implementation is under the elliptic curve, including the operations of zero-knowledge proofs. The elliptic curve is represented as $y^2 = x^3 + x$ and defined over F_p, with an embedding degree of $k = 6$. For the Type-A pairing, we set the prime subgroup order p to 256 bits and the base-field prime q to 1536 bits. The pairing parameters are generated using the JPBC configuration `TypeACurveGenerator(256,1536)`. In the resulting JPBC parameter file, the subgroup order is denoted by r, which corresponds to p here, while q denotes the base-field prime. The entire E-DAC is executed on the experimental platform. We gather the execution cost of each party in every phase of E-DAC on the platform. The results are presented in the fifth column of Table 3.2. This cost is the real execution time cost of E-DAC on the platform, covering communication expenses, and other associated processing costs on the devices.

Meanwhile, we conducted a comparative analysis of the computational overhead between our proposed E-DAC and the existing schemes, APECS [40] and AADEC [41]. The results of this comparison are illustrated in Fig. 3.2. During edge server registration, E-DAC exhibits high efficiency in authorization key generation since it only involves access key



(a) Communication Overhead for Edge Service Credentials (b) Communication Overhead for Service Access Credentials (c) Communication Overhead for Service Request

Figure 3.3: Communication Overhead Comparison of E-DAC, APECS, and AADEC

aggregation and edge share generation performed by the service provider. In contrast, both APECS and AADEC rely on computationally expensive ABE. Fig. 3.2(a) demonstrates that E-DAC outperforms APECS and AADEC in edge server registration as the number of authorized services increases. Fig. 3.2(b) shows the performance of the service provider to generate credentials for registering users, and E-DAC is the most efficient one because it only requires several polynomial evaluation operations, which are less time-consuming than the exponentiation operations required in APECS and AADEC for creating attribute keys in ABE. For generating the service requests in Fig. 3.2(c), a user needs to perform several exponentiation and bilinear pairing operations in E-DAC. This results in a heavier cost for our E-DAC than APECS, but E-DAC is still more efficient than AADEC. This overhead will not overwhelm the user device, as this operation only occurs once when the user needs to access a service. For handling concurrent service requests from users, we compare the time cost of service response for the edge server, as depicted in Fig. 3.2(d). Notably, with an increasing number of concurrent service requests, E-DAC demonstrates lower time cost on average for the edge server compared to APECS and AADEC. Additionally, Fig. 3.2(e) and Fig. 3.2(f) present the time cost for the service provider to revoke a user and an edge server in E-DAC, APECS, and AADEC, respectively. Both E-DAC and APECS are very efficient in user revocation. With an increasing number of edge servers, in E-DAC, when an edge server is revoked, the service provider does not need to update the authorization keys of other edge servers. Conversely, in APECS and AADEC, the service provider must re-key other edge servers. Therefore, E-DAC proves to be more efficient than APECS and AADEC in edge server revocation.

Furthermore, we present the communication overhead between the service provider, the edge server, and the user in fourth column of Table 3.2. It is worth noting that the

binary sizes of identity (ID_e or ID_u), service index, expiration time, and timestamp are all 64 bits. One notable feature of our E-DAC is that the edge service credential size remains constant, as shown in Fig. 3.3(a), thanks to the aggregation of the access keys of the services. On the contrary, in APECS and AADEC, the size of the edge service credential increases linearly with the number of authorized services. Figures 3.3(b) and 3.3(c) illustrate the comparison of the size of service access credentials and service requests, respectively. Our E-DAC demonstrates shorter sizes for both service access credentials and service requests compared to AADEC, albeit slightly longer than APECS. The main reason is the secret sharing mechanism employed in creating service access credentials for users. While efficient in computation, it imposes a slightly higher burden on communication. In addition, although AADEC is less communication and computational efficient than E-DAC and APECS, it achieves user anonymity and auditability, which are not supported by E-DAC and APECS.

3.6 Summary

In this chapter, we have proposed an efficient and distributed access control framework (E-DAC) tailored for dynamic services in the MEC environment. By aggregating access keys of various services, E-DAC enhances flexibility and efficiency in service authorization. The incorporation of 0-RTT authentication significantly reduces network bandwidth costs and minimizes communication rounds between users and edge servers. Moreover, E-DAC achieves rapid mutual authentication without relying on a centralized authentication server, making it suitable for scenarios where user devices have short-lived connections to edge servers due to high mobility. Additionally, E-DAC supports the dynamics of edge servers through efficient service authorization and low-cost service revocation. Therefore, E-DAC is particularly suitable for the real-time applications, where frequent computations and low latency are critical, such as autonomous driving, industrial IoT, and AR/VR.

Chapter 4

SecureEdge: Secure Data Integrity Verification for Mobile Edge Computing

4.1 Introduction

MEC is designed to manage end-user data at the network edge, reducing the need for long-distance data transmission and enabling near real-time service responses [111]. However, data stored and processed on edge nodes is highly susceptible to a range of cyber threats that can compromise its confidentiality, integrity, and availability [112, 113]. This vulnerability is particularly concerning because each edge node typically serves multiple users within its coverage area, caching diverse service data while lacking the robust security infrastructure of centralized cloud systems. As a result, end-user data may be corrupted by external attacks or rendered inaccessible due to operational failures at the edge node. Moreover, the limited storage capacity of edge devices [114, 115] exacerbates the problem that service data may be silently overwritten or deleted to accommodate new users' data, potentially without the original user's awareness [116]. These issues are serious in the pay-as-you-use edge storage services. These challenges underscore the critical need for efficient and reliable mechanisms to verify the integrity of service data stored at the edge, ensuring the delivery of secure and trustworthy services to end users.

However, verifying the integrity of users' data on edge nodes is non-trivial due to several key challenges. First, end devices typically do not retain a local copy of the service data

once it is uploaded to edge nodes. This limits the effectiveness of traditional integrity verification techniques, such as MAC, hash functions, or digital signatures, which rely on the original data to validate authentication tags. Second, both end devices and edge nodes are often resource-constrained, making it impractical to deploy computation-intensive schemes, such as publicly verifiable cloud data integrity protocols, in MEC [3]. Additionally, the mobility of end devices, such as vehicles, smartphones, or drones, means users may need to retrieve their data after moving outside the coverage area of the original edge node [117]. To ensure continuous and reliable data access, multi-replica storage becomes essential, enabling users to maintain multiple data copies across different edge servers to enhance availability [118]. In addition, service data may be shared among multiple users within the coverage of various edge nodes, adding another layer of complexity to ensuring consistent data integrity and availability across distributed environments [119, 120].

In this chapter, we propose SecureEdge, a novel framework of secure and efficient data integrity verification for MEC that addresses three key challenges: data-independent verification, computational efficiency, and multi-replica distributed verification. Inspired by the concept of Proof of Retrievability (PoR) [121], SecureEdge provides an efficient data-independent verification method, i.e, verifying the integrity of users' data maintained on multiple edge servers without retrieving the data copies from edge servers. It supports both private and public verifiability, enabling data owners (i.e., end users who generate and upload data to edge servers) to perform low-cost integrity verification in the multiplicative group, while also allowing TPAs to initiate independent public verifications in the bilinear group. By leveraging lightweight cryptographic mechanisms such as MACs and hash functions, private verifiability ensures high efficiency in tag generation and proof verification, while public verifiability promotes transparency by enabling third-party integrity verification, thereby preserving the trustworthiness of the data integrity verification results.

To enhance continuous data access and secure data sharing, SecureEdge supports multi-replica secure storage, where user data is distributed across multiple edge servers. However, if all servers store identical data copies, SecureEdge becomes vulnerable to edge-collision attacks, in which a dishonest server, lacking a valid replica, can leverage the same data stored on another server to fraudulently pass integrity verification. To counter this, SecureEdge generates distinct randomized replicas for each server by embedding controlled random noise into the data. Despite the differences, all replicas remain verifiable using a common authentication tag, enabling consistent integrity proofs. This randomization approach not only supports reliable integrity verification across distributed replicas, but also preserves user privacy, preventing both edge servers and TPAs from accessing the original data content. This privacy preservation is particularly important in publicly verifiable data integrity schemes, as such schemes are susceptible to privacy extraction attacks, where TPAs may

infer sensitive user data from the integrity proofs generated by edge servers. Additionally, SecureEdge can be extended to support data dynamics by integrating a sequence-enforced Merkle Hash Tree, allowing secure and efficient verification during operations such as insertions, deletions, and modifications. This is a critical capability in MEC environments, where user data is frequently updated in response to real-time service demands.

The main contributions of this work are fourfold:

- We propose SecureEdge, a distributed data integrity verification framework for MEC that enables both data owners and TPAs to verify the correctness of service data stored on edge nodes, mitigating risks from malicious attacks and operational failures.
- We design a secure multi-replica verification mechanism that prevents colluding edge nodes from passing integrity checks without maintaining valid data, while enhancing data availability and privacy preservation through distributed replica storage across MEC environments.
- We extend SecureEdge to support data dynamics, enabling secure and efficient integrity verification during real-time data operations, such as insertions, deletions, and modifications, critical for real-time and responsive MEC services.
- We provide a comprehensive security analysis to demonstrate that SecureEdge achieves security in both private and public verification, and we evaluate its performance, showing high computational efficiency in tag generation and proof verification for data owners, and in proof generation for edge servers.

4.2 Models and Design Goals

In this section, we present the entities in the system model, the security model, and our design goals.

4.2.1 System Model

The system model of data integrity verification in MEC environment is shown in Fig. 4.1, which consists of four entities: the cloud server, the data owner, the edge server, and the data user.

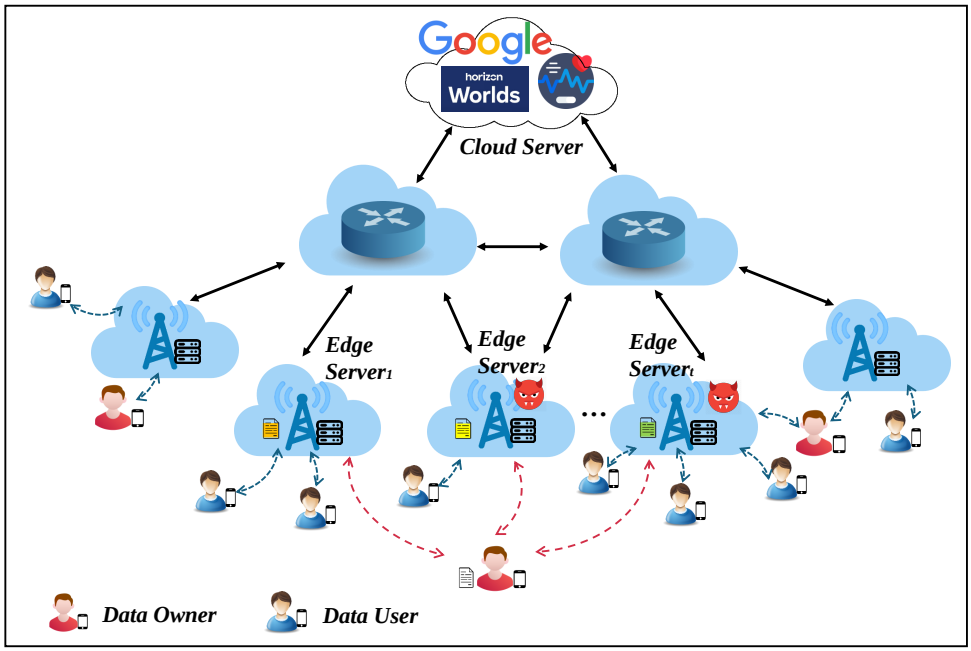


Figure 4.1: System Model

- **Cloud Server**: A cloud server owned by a service provider delivers a range of computing and storage services to end users, including AR/VR, and edge storage. These services typically rely on input data from users, such as videos, audio, or images, and return processed results back to them. To serve a large number of users with low latency, the service provider may delegate authorized edge servers to deliver these services at the network edge for its registered users.

- **Edge Server**: An edge server (edge node), such as a base station or a local server, is typically deployed by edge service providers. It stores data on behalf of the data owner and remains ready to respond to data integrity verification requests by generating and providing the necessary proof to confirm the data's authenticity and completeness.

- **Data Owner**: A data owner owns end devices such as mobile phones, tablets, vehicles, and sensors to collect diverse data from the surrounding environment. To reduce local storage overhead or access interactive services, the data is uploaded on multiple edge nodes (edge servers), eliminating the need to maintain local copies. This approach also enhances data redundancy, ensuring availability in the event of single points of failure, and enables data access and utilization by users within the

coverage area of these edge servers. However, outsourcing data to edge servers loses the direct control over the data for the data owner. Therefore, to prevent data corruption or loss, the data owner must regularly verify the integrity of the stored data and ensure it remains correctly maintained on the edge servers.

- **Data User**: A data user accesses cellular networks or services at the network edge using their end devices. If they are within the coverage area of edge servers and have proper authorization, they can access data stored by a data owner on those servers. To ensure data correctness, the data user also has the demand to verify the integrity of the accessed data owned by the data owners.

Additionally, a third party, such as a TPA, can be involved in verifying the integrity of edge data for auditing purposes. In our system model, data owners, data users, and third parties are all capable of acting as *verifiers* to perform edge data integrity verification. Therefore, the system supports both private verifiability and public verifiability. In the case of private verifiability, the goal is to enable data owners to efficiently verify the integrity of their outsourced data using their secret keys with minimal computational overhead. In contrast, public verifiability allows any party, including data owners, data users, and TPAs, to verify the integrity of the data without requiring access to the data owners' secret keys. This dual-mode verifiability enhances the flexibility and transparency of the system, ensuring both user data control and auditability.

The cloud server initiates the services and it does not directly involve in the edge data integrity verification. If needed, it can serve as the TPA to conduct verification. Data users can verify the integrity of data on edge servers. They can perform data possession verification directly and retain the verification results. Since data users can also act as third parties from the data owner's perspective, we refer to the verifier in public verification as the TPA for simplicity. In the private verification, the verifier is the data owner.

The edge serves maintain the data uploaded by the data owners. The number of edge servers required for storage should be dynamically determined according to the data volume, access latency, network conditions, and desired reliability level. At a minimum, the selected servers must collectively provide sufficient capacity for the data and its required replicas or coded fragments. Increasing the number of servers can improve availability, fault tolerance, and geographic accessibility, but also introduces additional communication, coordination, and maintenance overhead. Therefore, the system should select the smallest server set that satisfies the storage-capacity, latency, and reliability requirements, while dynamically adding or releasing servers as workloads and network conditions change. The selection methods of edge servers have been well studied [122, 123].

The multi-replica data integrity verification for MEC consists of six polynomial-time algorithms, namely, Setup, KeyGen, TagGen, Chal, Proof, and VeriP.

- Setup. This is a probabilistic algorithm run by the cloud server. It takes the security parameter κ as input and outputs the system parameter pp.
- KeyGen. This is a probabilistic algorithm run by the data owner. It takes the system parameter pp as input and generates the pair of the private key sk and the public key pk.
- TagGen. This is a probabilistic algorithm run by the data owner. It takes the system parameter pp, the user file M, the file name F_{id}, and the data owner's secret key sk as inputs, and outputs the authentication tags σ_i for each $i = 1$ to n, where n is the number of blocks into which the file M is divided. The file M is randomized to generate multiple replicas for all the edge servers that store M. Suppose the set of edge servers is $E = \{E_1, \cdots, E_t\}$, and the replica M^u is constructed for the edge server E_u, where $u \in \{1, t\}$. The replica M^u and the authentication tags $\{\sigma_1, \cdots, \sigma_n\}$ are uploaded to E_u.
- Chal. This is a probabilistic algorithm run by the verifier. It takes the system parameter pp and the file name F_{id} as inputs, and outputs a challenge $chal$ to verify the integrity of the file M on the edge servers.
- Proof. This is a probabilistic algorithm run by all challenged edge servers. A challenged edge server E_u takes the system parameter pp, the file name F_{id}, the challenged data blocks of the file M^u, and the authentication tags $\{\sigma_1, \cdots, \sigma_n\}$ as inputs, and outputs a data possession proof P^u of the challenged blocks.
- VeriP. This is a deterministic algorithm run by the verifier. It takes the system parameter pp, the file name F_{id}, and all the proof $P = \{P^1, \cdots, P^t\}$ from all the challenged edge servers as inputs, and outputs "success" to indicate the file M is intact, or "failure" to indicate the file is corrupted.

The correctness guarantees that the proof P should succeed data integrity verification if all the replicas of the file M are well maintained on the edge servers E. The correctness is also called completeness.

4.2.2 Security Model

In a MEC environment, each entity operates with its own obligations and incentives. End users, including data owners and data users, benefit from edge services and expect to be treated fairly by the edge servers. They are fully trusted to conduct data outsourcing, access, and processing. To preserve the trustworthiness of the data integrity verification results, public verifiability is essential. Edge servers, however, may face the following security threats:

- **Unpredicted Failures**: Edge servers may experience unforeseen hardware or software failures, such as drive malfunctions or power outages, potentially leading to data loss. They may also be vulnerable to unexpected external cyberattacks, including data tampering and deletion.
- **Malicious Behaviors**: Edge servers are assumed to be untrusted. We consider scenarios where an edge server may intentionally delete infrequently accessed data to free up storage for other data, driven by economic incentives. When the verifier later requests an integrity check, the edge server may attempt to forge a proof to falsely claim that the data remains intact. Furthermore, in cases where data is corrupted due to failures or external attacks, the edge server may still try to generate fraudulent proofs, such as using the intact replicas on another edge server, to pass integrity verifications and preserve its reputation.

According to the identified threats, the security of a multi-replica data integrity verification scheme is characterized by soundness, which ensures that a dishonest edge server cannot convince a verifier that it is storing a file unless it actually possesses the entire file.

Soundness: We formalize the soundness property of a multi-replica data integrity verification scheme in MEC, where the adversary $\mathcal{A}$ models a potentially untrusted edge server, and the challenger $\mathcal{C}$ represents the data owner. The soundness property ensures that an adversary cannot produce a valid proof of data possession without actually storing all required blocks of a file, except with a negligible probability (i.e., unless it can guess all missing blocks correctly).

- Setup: $\mathcal{C}$ executes the Setup algorithm with a security parameter κ to generate system parameters pp, which are sent to $\mathcal{A}$.
- Queries: $\mathcal{A}$ may issue adaptive queries to obtain data replicas and authentication tags.

– ReplicaGen Queries: $\mathcal{A}$ requests replicas of any chosen file M for any edge server E_u. The challenger runs the TagGen algorithm and returns the replica M^u.

– TagGen Queries: $\mathcal{A}$ requests the authentication tags of a file M. The challenger executes the TagGen algorithm and returns the tags.

- Proof: $\mathcal{A}$ assumes the role of the prover, generating a data integrity proof, while $\mathcal{C}$ acts as the verifier to simulate the process of data integrity verification. For a file M for which both TagGen and ReplicaGen queries have been issued, $\mathcal{A}$ runs the Chal and Proof algorithms to generate a challenge $chal$ and a corresponding proof P^u. The proof P^u is constructed based on the replica M^u (obtained through the ReplicaGen query) and the authentication tags (obtained through the TagGen query). The tuple $(chal, P^u)$ is then sent to $\mathcal{C}$, who verifies the validity of P^u using the VeriP algorithm and returns the verification result to $\mathcal{A}$.

- Output: Finally, $\mathcal{A}$ selects a file M^* provided that M^* should not have been queried in TagGen queries and ReplicaGen queries. $\mathcal{A}$ generates and returns a data integrity proof P^*.

If the cheating prover can generate P^*, so that it correctly answers an ϵ fraction of challenges, we say the prover is ϵ-admissible. Formally, $\Pr[\mathsf{VeriP}(M^*, chal^*, P^*) \rightleftharpoons \mathsf{Proof}(M^*, chal^*) = \text{“}success\text{”}] \geq \epsilon$. $\mathcal{A}$ wins the soundness game if it can successfully behave as an ϵ-admissible prover.

Definition 1. A multi-replica data integrity verification scheme is ϵ-sound if there exists a polynomial-time extraction algorithm such that, for any probabilistic polynomial-time adversary $\mathcal{A}$, whenever $\mathcal{A}$ behaves as an ϵ-admissible prover on a file M in the soundness game, the extraction algorithm can recover the complete file M^* from the proof P^*, except with a negligible probability.

Privacy against TPA: Since the TPA may act honestly but remain curious about data content, it is critical to guarantee that no sensitive information about a data owner's edge data is leaked to the TPA during the integrity verification process. Specifically, the TPA should not learn any additional information about the edge data beyond what is publicly available Now we formalize the model of the privacy against TPA by using a simulator as follows.

Definition 2. A multi-replica data integrity verification scheme achieves privacy against TPA if there exists a polynomial-time non-interactive simulator $\mathcal{S}$ for each curious verifier TPA*, such that the following random variables are computationally indistinguishable:

- $View_{TPA^*}(EID_u, chal, M^1, \cdots, M^t, F_{id}, tags, P^1, \cdots, P^t)$, where S is the random coins the scheme uses.
- $\mathcal{S}(F_{id}, pk, chal)$.

This definition implies that the simulator $\mathcal{S}$, which only has access to public inputs, can generate a simulated view that is computationally indistinguishable from the real view obtained by the adversarial verifier TPA* during its interaction with the edge server. This guarantees that TPA* learns nothing about the actual edge data beyond what is already publicly available.

4.2.3 Design Goals

We summarize that the desirable properties of SecureEdge include:

- **Correctness**: If an edge server correctly maintains the data of the data owner, it can generate a valid data integrity proof in response to a challenge, thereby successfully passing the data integrity verification.
- **Soundness**: If the edge data of the data owner has been corrupted or tampered with, the edge server cannot generate a valid proof that can pass the data integrity verification, even if it colludes with an honest edge server that correctly maintains the challenged data. Collusion resistance ensures that even the cooperation of multiple edge servers cannot generate a valid proof in response to a challenge unless all the required data blocks are intact and correctly stored.
- **Secure Multi-Replica Storage**: The data owner takes the original data file, generates multiple replicas through randomization, and securely stores one replica per edge server.
- **Privacy Preservation**: No information about a data owner's edge data is leaked to the TPA during the public integrity verification process.
- **Data Dynamics**: The data integrity verification scheme supports dynamic operations on the data files stored on edge servers, including modification, insertion, and deletion of data blocks.
- **Computational Efficiency**: The data integrity verification scheme ensures computational efficiency in both tag generation and proof verification for data owners under private verification, as well as in the proof generation process carried out by edge servers.

4.3 SecureEdge

In this section, we introduce SecureEdge, the secure multi-replica integrity verification framework, and extend SecureEdge to enable data dynamics.

4.3.1 Overview of SecureEdge

Inspired by the PoR [121], SecureEdge adopts lightweight cryptographic primitives, such as pseudo-random functions and hash functions, to construct an efficient data integrity verification mechanism for edge data. Data owners can efficiently generate authentication tags from data blocks and upload both the data and corresponding tags to edge servers. During integrity verification, SecureEdge retains key properties of PoR, including random sampling, homomorphic aggregation, and batch verification, to minimize computational and communication overhead. However, SecureEdge introduces multi-replica storage via randomization. Specifically, the original file is randomized to generate distinct replicas for different edge servers, thereby enhancing data availability while ensuring collusion resistance. So that, an edge server that does not store an intact replica of the challenged data cannot collude with another server that does in order to forge a valid proof. In SecureEdge, the authentication tags are generated from the original data blocks and remain consistent across replicas. Although the data replicas are randomized for distribution, the corresponding tags are unified and reusable for verification across all edge servers. Moreover, SecureEdge supports both private verifiability and public verifiability. Using a unified tag generation algorithm (TagGen) and a proof generation algorithm (Proof), different verifiers can assess data integrity through distinct modes. In private verification, data owners use their secret keys to verify proofs efficiently in the multiplicative group, enabling lightweight verification via the VeriP algorithm suitable for resource-constrained devices. In public verification, TPAs verify proofs using only the data owner's public key in the bilinear group, thereby enabling public auditability of edge-stored data without revealing its contents.

Replica security considers both localized and distributed replica deployment. In localized replication, multiple replicas are stored on the same edge server to improve local availability; however, a dishonest server may retain only one copy and use it to generate proofs for all claimed replicas. To prevent this replica-replacement attack, each replica is independently randomized and authenticated so that a valid proof for one replica cannot be reused for another. In distributed replication, replicas are placed on different edge servers to tolerate individual server failures and improve geographical availability. The verification mechanism checks whether each designated server possesses its assigned replica and

prevents one server from using another server's copy to falsely satisfy an audit. Thus, the scheme guarantees replica distinctness, possession, and integrity at the logical level. It does not, however, guarantee physical separation when multiple virtual servers are controlled by the same provider, nor can it prevent fully colluding servers from temporarily sharing replicas unless location proofs, timing constraints, or trusted hardware are additionally employed.

4.3.2 Detailed SecureEdge

We describe the proposed SecureEdge in detail. The notations of SecureEdge are presented in Table 4.1.

4.3.2.1 Setup

This algorithm bootstraps the whole system. Given the security parameter κ, let p be a large prime. $\hat{e} : G \times G \rightarrow G_T$ is a bilinear map. G and G_T are cyclic groups of the prime order p. g is a generator of G. $H : \{0,1\}^* \stackrel{R}{\rightarrow} Z_p^*$ be a hash function. $Enc_k()$ is a symmetric encryption algorithm with the key k and the corresponding decryption algorithm is $Dec_k()$. $MAC_k() \rightarrow Z_p^*$ is an MAC algorithm with the key k, and $SSig_k()$ is a signature algorithm with the key k. $\phi_k() \rightarrow Z_p^*$ is the Pseudo-Random Function (PRF) with a key k, $\pi_k() \rightarrow the\ index\ set\ of\ edge\ servers$ is a Psudo-Ramdom Permutation (PRP) with the key k, and $\lambda_k() \rightarrow the\ index\ set\ of\ data\ blocks$ is a PRP function with the key k. The system parameter $pp = (p, G, G_T, \hat{e}, H, Enc_k(), Dec_k(), MAC_k(), SSig_k(), \phi_k(), \pi_k(), \lambda_k())$.

4.3.2.2 KeyGen

Given a security parameter, a data owner DO generates a signing keypair (spk, ssk), a symmetric encryption key k_{enc}, and a random MAC key k_{mac}. DO also randomly chooses $x \stackrel{R}{\leftarrow} Z_p^*$ and computes $S = g^x$. The secret key is $sk = \{ssk, k_{enc}, k_{mac}, x\}$, and the public key is $pk = \{spk, S\}$.

4.3.2.3 TagGen

Given the file M with the file identity F_{ID}, DO applies the erasure code to obtain M', then splits M' into n blocks, and each block is further divided into s sectors. The processed file

Table 4.1: Notations of SecureEdge

Symbol	Description
E_u	Edge server u
EID_u	Identity of edge server E_u
$\mathsf{Enc}_k, \mathsf{Dec}_k$	Symmetric encryption and decryption algorithms
MAC_k	Message authentication code algorithm
SSig_k	Digital signature algorithm
FID	File identity
n	Number of data blocks in a file
s	Number of sectors in each data block
x_j	Random number associated with sector j for authentication tag generation
m_{ij}	Sector j of the original data block i
M^u	Randomized file replica assigned to edge server E_u
m^u_{ij}	Sector j of replicated block i stored by E_u
t	Number of edge servers and file replicas
y_u	Random number for replica of E_u
r^u_j	Random masking value for sector j of replica of E_u
σ_i	Authentication tag of data block i
$\boldsymbol{\sigma}$	Set of authentication tags
w_0, w_1	Private and public verification metadata, respectively
τ_0, τ_1	Authenticated private and public verification information
e	Number of edge servers selected for an integrity challenge
T	Index set of challenged edge servers
E_{chal}	Identity set of challenged edge servers
c	Number of data blocks sampled in a challenge
I	Index set of challenged data blocks
ν_i	Random challenge coefficient for block i
Q	Challenge set, $Q = \{(i, \nu_i) : i \in I\}$
μ^u_j	Aggregated value of challenged sectors for replica of E_u
σ^u	Aggregated authentication tag generated by E_u
P^u	Proof generated by E_u

M^* is $\{m_{ij}\}_{1\leq i\leq n, 1\leq j\leq s}$, $m_{ij} \in Z_p^*$. *DO* chooses s random numbers $x_1, \cdots, x_s \overset{R}{\leftarrow} Z_p^*$ and computes $\{S_1, \cdots, S_s\} = \{g^{x_1}, \cdots, g^{x_s}\}$. For each $i, 1 \leq i \leq n$, *DO* computes the tag of the block i:

$$\sigma_i = xH(F_{ID}||i) + \sum_{j=1}^{s} x_j m_{ij}. \tag{4.1}$$

where $||$ is the concatenation of two elements.

The file is randomized before uploading to generate t different replicas, which are uploaded to the t edge servers $E = \{E_1, \cdots, E_t\}$ with the identities $EID_u, 1 \leq u \leq t$, along with their authentication tags $\sigma = \{\sigma_i\}, 1 \leq i \leq n$. To generate t different replicas, *DO* selects t elements $y_1, \cdots, y_t \overset{R}{\leftarrow} Z_p^*$ and computes $\{Y_1, \cdots, Y_t\} = \{g^{y_1}, \cdots, g^{y_t}\}$. The replica for E_u, $1 \leq u \leq t$, is $\{m_{ij}^u\}_{1\leq i\leq n, 1\leq j\leq s}$, here $m_{ij}^u = m_{ij} + r_j^u$, where $r_j^u = y_u H(EID_u||j)$. Here s random values are generated for each edge server E_u, and for all t edge servers, an independent set of s random values are used to generate t replicas, one replica for an edge server. *DO* also computes $\{R_1^u, \cdots, R_s^u\} = \{g^{r_1^u}, \cdots, g^{r_s^u}\}$.

Then, let $w_0 = Enc_{k_{enc}}(n||s||x_1||\cdots||x_s||y_1||\cdots||y_t)$ and $w_1 = n||s||S||S_1||\cdots||S_s||Y_1||\cdots||Y_t||R_1^u||\cdots||R_s^u$. *DO* generates $\tau_0 = w_0||MAC_{k_{mac}}(w_0)$ and $\tau_1 = w_1||SSig_{ssk}(w_1)$. *DO* uploads the file $F = \{FID, \tau_0, \tau_1, \{m_{ij}^u\}_{1\leq i\leq n, 1\leq j\leq s}\}$, along with the tags $\sigma = \{\sigma_i\}, 1 \leq i \leq n$ to corresponding edge servers in the set E. *DO* also records the set of edge servers that stores the file F.

4.3.2.4 Chal

A verifier, including a data owner, or a TPA, generates a challenge *chal* to validate the integrity of the file M on the edge servers. e is the number of the challenged edge servers. The verifier picks e-element subset T of the set $[1, t]$ by the PRP $\pi_a(i)$, $1 \leq i \leq e$ with the key $a \overset{R}{\leftarrow} Z_p^*$. The group of the challenged edge servers is denoted as $E_{chal} = \{EID_u\}$, $u \in T$. c is the number of blocks sampled for integrity verification. The verifier picks c-element subset I of the set $[1, n]$ by the PRP $\lambda_b(i)$, $1 \leq i \leq c$ with the key $b \overset{R}{\leftarrow} Z_p^*$. For each $i \in I$, a random number $\nu_i = \phi_\nu(i)$, the PRF key $\nu \overset{R}{\leftarrow} Z_p^*$. Let Q be the set $\{(i, \nu_i)\}$, $i \in I$. The challenge is denoted as $chal = \{e, T, E_{chal}, F_{ID}, c, Q\}$ and sent to all edge servers in the set E.

4.3.2.5 Proof

Once receiving the challenge *chal*, an edge server E_u, where u represents the index of the edge server E_u, first checks whether it is challenged by the verifier. E_u parses the challenge as $\{e,\ T,\ E_{chal},\ F_{ID},\ c,\ Q\}$. E_u checks whether its identity is in the set E_{chal}. If not, E_u ignores the challenge. Otherwise, E_u generates and responds a proof P^u to prove the integrity of the file M. With the file identity F_{ID}, E_u firstly finds the file replica $\{m_{ij}^u\}$, $1 \leq i \leq n$, $1 \leq j \leq s$, With $Q = \{(i, \nu_i)\}, i \in I$, for each i, the edge server E_u computes

$$\mu_j^u = \sum_{(i,\nu_i)\in Q} \nu_i m_{ij}^u, 1 \leq j \leq s. \tag{4.2}$$

$$\sigma^u = \sum_{(i,\nu_i)\in Q} \nu_i \sigma_i. \tag{4.3}$$

The edge server E_u sends the proof $P^u = \{EID_u, F_{ID}, \mu_1^u, \cdots, \mu_s^u, \sigma^u\}$ to the verifier in response.

4.3.2.6 VeriP

This algorithm could support the private verification of P^u for the data owner DO and the public verification of P^u for the TPA.

Private Verifiability: DO retrieves τ_0 based on F_{ID} and parses it as w_0 and its MAC. Parse sk as $\{ssk, k_{enc}, k_{mac}, x\}$. DO first uses k_{mac} to verify the MAC. If the MAC is invalid, reject by returning 'MAC failure' and halt. Otherwise, DO uses k_{enc} to decrypt w_0 and recover $\{n,\ s,\ x_1,\ \cdots,\ x_s,\ y_1,\ \cdots,\ y_t\}$.

DO parses the proof P^u from the edge server E_u as $\{EID_u,\ F_{ID},\ \mu_1^u,\ \cdots,\ \mu_s^u,\ \sigma^u\}$, and calculates $r_j^u = y_u H(EID_u||j)$, $1 \leq j \leq s$. Based on the set $Q = \{(i, \nu_i)\}$ generated in Chal phase, $i \in I$, DO verifies the validity of the proof P^u as:

$$\sigma^u = \sum_{(i,\nu_i)\in Q} \nu_i x H(F_{ID}||i) + \sum_{j=1}^{s} x_j \mu_j^u - \sum_{(i,\nu_i)\in Q} \nu_i \sum_{j=1}^{s} x_j r_j^u. \tag{4.4}$$

If the equation holds, the file replica on the edge server E_u is valid; Otherwise, the replica is corrupted.

The correctness of data integrity verification on E_u is given in Equ. (4.5).

$$
\begin{aligned}
\sigma^u &= \sum_{(i,\nu_i)\in Q} \nu_i \sigma_i \qquad (4.5)\\
&= \sum_{(i,\nu_i)\in Q} \nu_i (xH(F_{ID}||i) + \sum_{j=1}^{s} x_j m_{ij})\\
&= \sum_{(i,\nu_i)\in Q} \nu_i xH(F_{ID}||i) + \sum_{(i,\nu_i)\in Q} \nu_i \sum_{j=1}^{s} x_j (m_{ij}^u - r_j^u)\\
&= \sum_{(i,\nu_i)\in Q} \nu_i xH(F_{ID}||i) + \sum_{j=1}^{s} x_j \sum_{(i,\nu_i)\in Q} \nu_i m_{ij}^u - \sum_{(i,\nu_i)\in Q} \nu_i x_j r_j^u\\
&= \sum_{(i,\nu_i)\in Q} \nu_i xH(F_{ID}||i) + \sum_{j=1}^{s} x_j \mu_j^u - \sum_{(i,\nu_i)\in Q} \nu_i \sum_{j=1}^{s} x_j r_j^u.
\end{aligned}
$$

For batch verification of the proofs from all challenged edge servers $E_{chal} = \{EID_u\}$, $u \in T$, DO parses the proofs and calculates $r_j^u = y_u H(EID_u||j)$, $1 \le j \le s$, $u \in T$. Based on the set $Q = \{(i, \nu_i)\}$ generated in Chal phase, $i \in I$, DO verifies the validity of the proofs P^u, $u \in T$ as:

$$
\sum_{u\in T} \sigma^u = |T| \sum_{(i,\nu_i)\in Q} \nu_i xH(F_{ID}||i) + \sum_{j=1}^{s} x_j \sum_{u\in T} \mu_j^u - \sum_{(i,\nu_i)\in Q} \nu_i \sum_{j=1}^{s} x_j \sum_{u\in T} r_j^u. \qquad (4.6)
$$

If the equation holds, all the replicas on the edge servers $E_u, u \in T$ are valid; Otherwise, some replicas have been corrupted and DO individually checks each proof P^u to find the corrupted ones.

Public Verifiability: The TPA retrieves τ_1 based on F_{ID} and parses it as w_1 and its signature $SSig_{ssk}(w_1)$. Parse w_1 as $\{n, s, S, S_1, \cdots, S_s, Y_1, \cdots, Y_t, R_1^u, \cdots, R_s^u\}$. The TPA first uses spk to verify $SSig_{ssk}(w_1)$. If $SSig_{ssk}(w_1)$ is invalid, reject by returning 'Signature failure' and halt. Otherwise, the TPA parses the proof P^u from the edge server as $\{EID_u, F_{ID}, \mu_1^u, \cdots, \mu_s^u, \sigma^u\}$. Based on the public information in w_1, TPA verifies the validity of the proof P^u as:

$$
\hat{e}(g^{\sigma^u}, g) \cdot \prod_{j=1}^{s} \hat{e}(S_j^{\sum_{(i,\nu_i)\in Q} \nu_i}, R_j^u) = \hat{e}(S^{\sum_{(i,\nu_i)\in Q} \nu_i H(F_{ID}||i)}, g) \cdot \hat{e}(\prod_{j=1}^{s} S_j^{\mu_j^u}, g). \qquad (4.7)
$$

If the equation holds, the file copy on the edge server E_u is valid, otherwise, it is corrupted.

The correctness of data integrity verification on E_u is given in Equ. (4.8).

For batch verification of the proofs from all challenged edge servers $E_{chal} = \{EID_u\}$, $u \in T$, the TPA verifies the validity of the proofs P^u, $u \in T$ as

$$\hat{e}(g^{\sum_{u\in T}\sigma^u}, g) \cdot \prod_{j=1}^{s} \hat{e}(S_j^{\sum_{(i,\nu_i)\in Q}\nu_i}, \prod_{u\in T} R_j^u) = \hat{e}(S^{\sum_{(i,\nu_i)\in Q}^{|T|}\nu_i H(F_{ID}||i)}, g) \cdot \prod_{u\in T} \hat{e}(\prod_{j=1}^{s} S_j^{\mu_j^u}, g). \quad (4.9)$$

If the equation holds, all the replicas on the edge servers $E_u, u \in T$ are valid; Otherwise, some replicas have been corrupted and the TPA individually checks each proof P^u to find the corrupted ones.

4.3.3 Data Dynamics

Here, we extend SecureEdge to enable data dynamics, supporting updated data integrity verification during data insertion, deletion, and modification on edge servers.

A MHT [124] is a well-studied authentication structure that can efficiently and securely verify the integrity of data elements. It is a binary tree in which the leaves are the hash values of authenticated data elements, and the internal nodes are the hash values of their left and right children. The root of the MHT serves as the tag used to verify the correctness and sequence of all data elements in the tree. As shown in Fig. 4.2(a), for authenticating x_1, x_5, the prover sends the verifier the Auxiliary Authentication Information (AAI) $\Omega_1 = \langle h(x_2), h_D \rangle$ and $\Omega_5 = \langle h(x_6), h_F \rangle$. The verifier computes $h(x_1)$, $h(x_5)$, $h_C = h(h(x_1)||h(x_2))$, $h_E = h(h(x_5)||h(x_6))$, $h_A = h(h_C||h_D)$, $h_B = h(h_E||h_F)$, and $h_R = h(h_A||h_B)$, and checks whether h_R matches the tag. Due to this capability, MHT has been widely used to authenticate both the values and the positions of data blocks in cloud data integrity verification. In this work, we further extend the MHT into a sequence-enforced MHT (sMHT), which more efficiently manages all data blocks. Specifically, the number of leaves that a node can reach is hashed along with the data. This number, representing the number of leaves under a node, also defines the rank of its children. As illustrated in Fig. 4.2(b), the leaf node x_1 is hashed as $h(x_1||1)$, the leaf node x_2 as $h(x_2||1)$, and their parent node as $h_C = h(h(x_1||1)||h(x_2||1)||2)$. The root node is $h_R = h(h_A||h_B||8)$. To verify x_3's value and position, we use the root $(h_R, 8)$ and AAI $= (h(x_4||1), 1, 0)$, $(h_C, 2, 1)$, $(h_B, 4, 2)$ to compute: $h_D = h(h(x_3||1)||h(x_4||1)||(1+1))$, $h_A = h(h_C||h_D||(2+2))$, and $h_{R'} = h(h_A||h_B||(4+4))$, then verify if $h_R = h_{R'}$. Each AAI entry is a three-tuple (A, B, C) representing the hash value, the number of leaves, and the

$$\hat{e}(g^{\sigma^u}, g) \cdot \prod_{j=1}^{s} \hat{e}(S_j^{\sum_{(i,\nu_i)\in Q} \nu_i}, R_j^u) \tag{4.8}$$

$$= \hat{e}(g^{\sum_{(i,\nu_i)\in Q} \nu_i \sigma_i}, g) \cdot \prod_{j=1}^{s} \hat{e}(S_j^{\sum_{(i,\nu_i)\in Q} \nu_i}, R_j^u)$$

$$= \hat{e}(g^{\sum_{(i,\nu_i)\in Q} \nu_i (xH(F_{ID}||i) + \sum_{j=1}^{s} x_j m_{ij})}, g) \cdot \prod_{j=1}^{s} \hat{e}(S_j^{\sum_{(i,\nu_i)\in Q} \nu_i}, R_j^u)$$

$$= \hat{e}(g^{\sum_{(i,\nu_i)\in Q} \nu_i xH(F_{ID}||i) + \sum_{(i,\nu_i)\in Q} \nu_i \sum_{j=1}^{s} x_j m_{ij}}, g) \cdot \prod_{j=1}^{s} \hat{e}(S_j^{\sum_{(i,\nu_i)\in Q} \nu_i}, R_j^u)$$

$$= \hat{e}(g^{\sum_{(i,\nu_i)\in Q} \nu_i xH(F_{ID}||i) + \sum_{(i,\nu_i)\in Q} \nu_i \sum_{j=1}^{s} x_j (m_{ij}^u - r_j^u)}, g) \cdot \prod_{j=1}^{s} \hat{e}(S_j^{\sum_{(i,\nu_i)\in Q} \nu_i}, R_j^u)$$

$$= \hat{e}(g^{\sum_{(i,\nu_i)\in Q} \nu_i xH(F_{ID}||i) + \sum_{(i,\nu_i)\in Q} \nu_i \sum_{j=1}^{s} x_j m_{ij}^u - \sum_{(i,\nu_i)\in Q} \nu_i \sum_{j=1}^{s} x_j r_j^u}, g) \cdot \prod_{j=1}^{s} \hat{e}(S_j^{\sum_{(i,\nu_i)\in Q} \nu_i}, R_j^u)$$

$$= \hat{e}(g^{\sum_{(i,\nu_i)\in Q} \nu_i xH(F_{ID}||i) + \sum_{j=1}^{s} x_j \mu_j^u - \sum_{(i,\nu_i)\in Q} \nu_i \sum_{j=1}^{s} x_j r_j^u}, g) \cdot \prod_{j=1}^{s} \hat{e}(S_j^{\sum_{(i,\nu_i)\in Q} \nu_i}, R_j^u)$$

$$= \hat{e}(g^{\sum_{(i,\nu_i)\in Q} \nu_i xH(F_{ID}||i)}, g) \cdot \hat{e}(g^{\sum_{j=1}^{s} x_j \mu_j^u}, g) \cdot \hat{e}(g^{-\sum_{(i,\nu_i)\in Q} \nu_i \sum_{j=1}^{s} x_j r_j^u}, g) \cdot \prod_{j=1}^{s} \hat{e}(S_j^{\sum_{(i,\nu_i)\in Q} \nu_i}, R_j^u)$$

$$= \hat{e}(S^{\sum_{(i,\nu_i)\in Q} \nu_i H(F_{ID}||i)}, g) \cdot \hat{e}(\prod_{j=1}^{s} S_j^{\mu_j^u}, g) \cdot \prod_{j=1}^{s} \hat{e}(g_j^{x_j \sum_{(i,\nu_i)\in Q} \nu_i}, g^{r_j^u})^{-1} \cdot \prod_{j=1}^{s} \hat{e}(S_j^{\sum_{(i,\nu_i)\in Q} \nu_i}, R_j^u)$$

$$= \hat{e}(S^{\sum_{(i,\nu_i)\in Q} \nu_i H(F_{ID}||i)}, g) \cdot \hat{e}(\prod_{j=1}^{s} S_j^{\mu_j^u}, g).$$

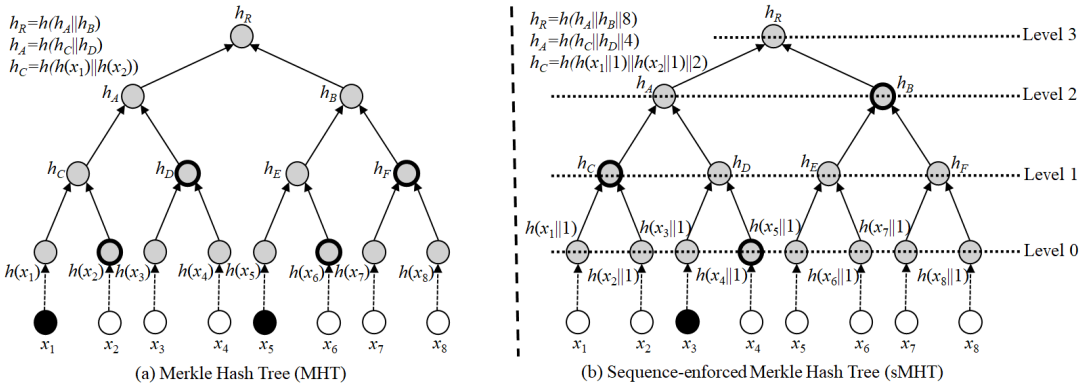


Figure 4.2: Example of MHT and sMHT

level, respectively. The sMHT is leveraged to support updates of authentication tags, enabling block modification, insertion, and deletion, while maintaining the sequence of data blocks and verifying the correctness of data replicas on edge servers.

Now we present how SecureEdge can efficiently handle fully dynamic data operations, including data modification, insertion, and deletion for edge data storage. To support data dynamics, the construction of authentication tags in SecureEdge must be modified, because the block index i in σ_i presents an obstacle to enabling dynamic operations. Therefore, to support data dynamics in SecureEdge, the block index i should be removed from σ_i. Thus, the authentication tag is altered to $\sigma_i^u = xH(F_{ID}||h(m_i^u)) + \sum_{j=1}^{s} x_j m_{ij}$. With the involvement of $h(m_i^u)$, σ_i^u becomes different for different edge servers in E, meaning that DO needs to generate a distinct σ_i^u on a block i for an edge server E_u, where $u \in \{1, \cdots, t\}$. This modification enables DO to update the data blocks on any edge server in E, without affecting the authentication tags of other edge servers. However, after removing the block index, SecureEdge is insecure because it becomes vulnerable to replace attacks in which arbitrary blocks and their authentication tags can replace the challenged blocks and tags to succeed data integrity verification. Therefore, we need to reconstruct the sequence of the blocks using sMHT. Here, we assume that the sMHT has been constructed for the file M by DO and is maintained on the edge servers. Now, we present how SecureEdge achieves dynamic data operations.

4.3.3.1 Block Modification

Suppose the data owner DO wants to modify the i-th block m_i to m_i'. Based on the new block m_i', which is divided into s sectors, DO generates the multiple replicas for each edge

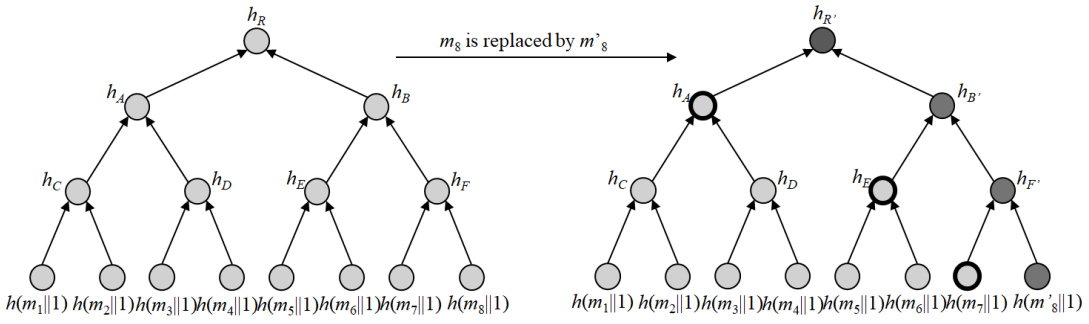


Figure 4.3: Example of sMHT Update under Block Modification Operation

server in E as $m_{ij}^{'u} = m'_{ij} + r_j^u$, where $r_j^u = y_u H(EID_u||j)$, for $1 \leq u \leq t$ and $1 \leq j \leq s$. The new block for E_u's replica is $m_i^{'u} = (m_{i1}^{'u}, \cdots, m_{is}^{'u})$. DO also generates the new authentication tag $\sigma_i^{'u} = xH(F_{ID}||h(m_i^{'u})) + \sum_{j=1}^{s} x_j m'_{ij}$. Then, DO sends the update message $update = \{Modify, i, m_i^{'u}, \sigma_i^{'u}\}$ to each edge server E_u, for $1 \leq u \leq t$. Upon receiving the request, the edge server E_u 1) replaces the block m_i^u with $m_i^{'u}$; 2) replaces the authentication tag σ_i^u with the new tag $\sigma_i^{'u}$; 3) replaces $h(m_i^u||1)$ with $h(m_i^{'u}||1)$ to update the sMHT and generates a new root $h_{R'}$, as shown in the example in Fig. 4.3. Finally, E_u returns DO a proof for the modification operation, $P_{update} = \{Modify, h(m_i^u||1), \Omega_i^u, Sign(x, h_R), h_{R'}\}$, where Ω_i^u is the AAI for $h(m_i^u||1)$. After receiving the proof, DO first generates h_R with $h(m_i^u||1)$ and Ω_i^u and verifies the validity of $Sign(x, h_R)$. If it is not valid, DO output failure; Otherwise, DO computes the new root with $h(m_i^{'u}||1)$ and Ω_i^u and compares with $h_{R'}$ to determine whether E_u has performed the modification operation correctly. If not, DO returns failure; Otherwise, DO signs the new root $h_{R'}$ with the private key x to generate the new root signature $Sign(x, h_{R'})$ and sends it to E_u. E_u replaces $Sign(x, h_R)$ with $Sign(x, h_{R'})$ for update. After all edge servers in E complete the update operation, DO executes the integrity verification operation to make sure that all the edge nodes in E have modified the block correctly and deletes the local data if the verification succeeds.

4.3.3.2 Block Insertion

Suppose the data owner DO wants to insert a new block m_* after the i-th block m_i. Based on the new block m_*, which is divided into s sectors, DO generates the multiple replicas for each edge server in E as $m_{*j}^u = m_{*j} + r_j^u$, where $r_j^u = y_u H(EID_u||j)$, for $1 \leq u \leq t$ and $1 \leq j \leq s$. The new block for EID_u's replica is $m_*^u = (m_{*1}^u, \cdots, m_{*s}^u)$. DO also generates the new authentication tag $\sigma_*^u = xH(F_{ID}||h(m_*^u)) + \sum_{j=1}^{s} x_j m_{*j}$. Then, DO sends the

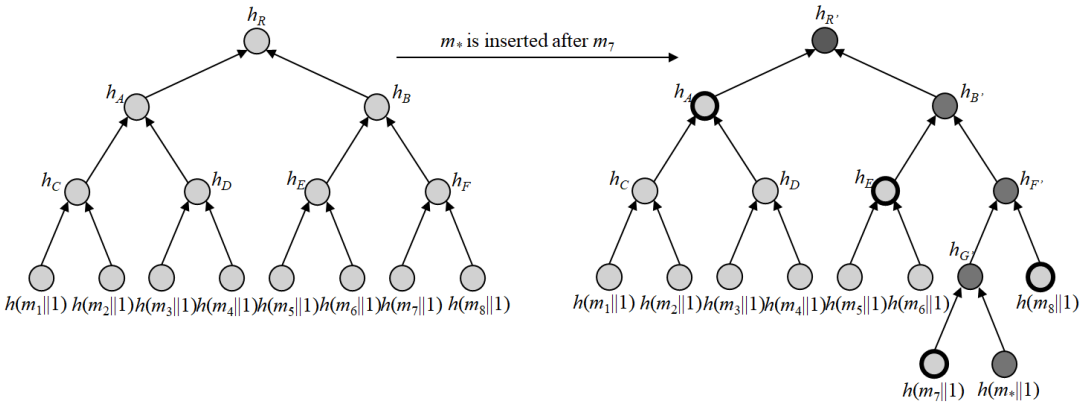


Figure 4.4: Example of sMHT Update under Block Insert Operation

update message $update = \{Insert, i, m_*^u, \sigma_*^u\}$ to each edge server E_u, for $1 \leq u \leq t$. Upon receiving the request, the edge server E_u 1) stores the new block m_*^u after the block m_i^u; 2) stores the authentication tag σ_*^u; 3) inserts $h(m_*^u||1)$ after $h(m_i^u||1)$ ($h(m_*^u||1)$ is the right child of $h(m_i^u||1)$'s parent) in the sMHT and generates a new root $h_{R'}$, as shown in the example in Fig. 4.4. Finally, E_u returns DO a proof for the insertion operation, $P_{update} = \{Insert, h(m_i^u||1), \Omega_i^u, Sign(x, h_R), h_{R'}\}$, where Ω_i^u is the AAI for $h(m_i^u||1)$. After receiving the proof, DO first generates h_R with $h(m_i^u||1)$ and Ω_i^u and verifies the validity of $Sign(x, h_R)$. If it is not valid, DO outputs failure; Otherwise, DO computes the new root with $h(m_*^u||1)$, $h(m_i^u||1)$, and Ω_i^u and compares with $h_{R'}$ to determine whether E_u has performed the insertion operation correctly. If not, DO returns failure; Otherwise, DO signs the new root $h_{R'}$ with the private key x to generate the new root signature $Sign(x, h_{R'})$ and sends it to E_u. E_u replaces $Sign(x, h_R)$ with $Sign(x, h_{R'})$ for update. After all edge servers in E complete the update operation, DO executes the integrity verification operation to make sure that all the edge nodes in E have inserted the new block correctly and deletes the local data if the verification is succeeded.

4.3.3.3 Block Deletion

Suppose the data owner DO wants to delete the i-th block m_i. DO sends the update message $update = \{Delete, i\}$ to each edge server E_u, for $1 \leq u \leq t$. Upon receiving the request, the edge server E_u 1) deletes the block m_i^u; 2) deletes the authentication tag σ_i^u; 3) deletes $h(m_i^u||1)$ in the sMHT and generates a new root $h_{R'}$, as shown in the example in Fig. 4.5. Finally, E_u returns DO a proof for the deletion operation,

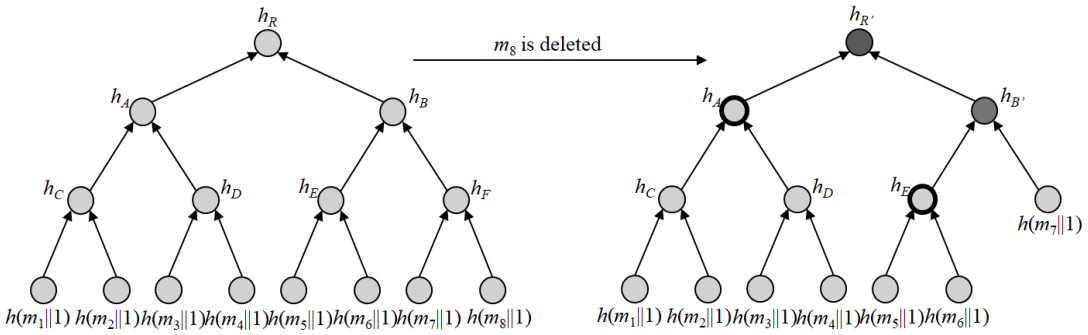


Figure 4.5: Example of sMHT Update under Block Deletion Operation

$P_{update} = \{Delete, h(m_i^u||1), \Omega_i^u, Sign(x, h_R), h_{R'}\}$, where Ω_i^u is the AAI for $h(m_i^u||1)$. After receiving the proof, DO first generates h_R with $h(m_i^u||1)$ and Ω_i^u and verifies the validity of $Sign(x, h_R)$. If it is not valid, DO outputs failure; Otherwise, DO computes the new root with $h(m_*^u||1)$ (suppose $h(m_*^u||1)$ is the sibling of $h(m_i^u||1)$) and other AAI in Ω_i^u and compares with $h_{R'}$ to determine whether E_u has performed the deletion operation correctly. If not, DO returns failure; Otherwise, DO signs the new root $h_{R'}$ with the private key x to generate the new root signature $Sign(x, h_{R'})$ and sends it to E_u. E_u replaces $Sign(x, h_R)$ with $Sign(x, h_{R'})$ for update. After all edge servers in E complete the update operation, DO executes the integrity verification operation to make sure that all the edge nodes in E have deleted the i-th block if the verification is succeeded.

4.4 Security Analysis

The soundness of SecureEdge relies on the difficulty to forge the tag σ_i, which is constructed from the hash function and the parameters, including $(x, x_1, \cdots, x_s)$. x is the secret key of DO and $(x_1, \cdots, x_s)$ are the random parameters, which are encrypted by the symmetric encryption algorithm and outsourced to the edge servers. Because SecureEdge supports both private verifiability and public verifiability, we prove the soundness of the privately verifiable SecureEdge and publicly verifiable SecureEdge separately.

Theorem 1. If the MAC scheme is unforgeable and the symmetric encryption algorithm is semantically secure, no adversary can defeat the soundness of privately verifiable SecureEdge with non-negligible probability, that is, the data owner will not accept data integrity proofs generated by an adversary without maintaining the challenged file correctly.

We can prove the theorem with a series of games.

Game 0. The first game, Game 0, is the challenge game of soundness in section 4.2.2, with the changes of private verifiability, in which the verifier is the data owner.

Game 1. Game 1 is the same as Game 0, with one difference. The challenger stores a list of tags, the MACs, and files. If an adversary generates an MAC $MAC_{k_{mac}}(w_0)$ that verifies as valid under k_{mac}, but $MAC_{k_{mac}}(w_0)$ is not generated by the challenger, the challenger aborts.

The difference between Game 0 and Game 1 is the unforgeability of the MAC algorithm. If the MAC algorithm is secure, no adversary can forge a valid MAC.

Game 2. Game 2 is the same as Game 1, with one difference. The challenger includes in the tags not the encryption of $n||s||x_1||\cdots||x_s||y_1||\cdots||y_t$, but a random string with the same length. If the adversary can distinguish the random string between $n||s||x_1||\cdots||x_s||$ $y_1||\cdots||y_t$, given the ciphertexts, the challenger declares failure and aborts.

The difference between Game 2 and Game 1 is the semantic security of the symmetric encryption algorithm. If no adversary can distinguish the random string between $n||s||x_1||\cdots||x_s||y_1||\cdots||y_t$, given the ciphertexts, assuming that the secret key k_{enc} is not disclosed, this algorithm is secure.

Game 3. Game 3 is the same as Game 2, with one difference. The challenger includes the tags that are generated by the data owner, and the corresponding blocks and replicas. If the adversary generates a valid $\tilde{\sigma}^u$, so that $\tilde{\sigma}^u$ is the valid proof of the challenged blocks, the challenger declares failure and aborts.

The difference between Game 3 and Game 2 is the unforgeability of the tags. If $\tilde{\sigma}^u$ is the valid proof, so that

$$\tilde{\sigma}^u = \sum_{(i,\nu_i)\in Q} \nu_i x H(F_{ID}||i) + \sum_{j=1}^{s} x_j \tilde{\mu}_j^u - \sum_{(i,\nu_i)\in Q} \nu_i \sum_{j=1}^{s} x_j \tilde{r}_j^u. \tag{4.10}$$

If Equ. 4.10 subtracts Equ. 4.4, we obtain

$$\tilde{\sigma}^u - \sigma^u = \sum_{j=1}^{s} x_j(\tilde{\mu}_j^u - \mu_j^u) + \sum_{(i,\nu_i)\in Q} \nu_i \sum_{j=1}^{s} x_j(r_j^u - \tilde{r}_j^u). \tag{4.11}$$

As r_j^u and $\tilde{r}_j^u$ are independent to the data blocks, they can be the same for the same file that is stored on the same edge server. Without the knowledge of $\{x_1, \cdots, x_s\}$, the probability that the remaining part equal is $1/p$, which is the negligible because p is a large prime. Therefore, the gap between Game 3 and Game 2 is negligible.

Suppose the probability that an adversary can forge a MAC is denoted by p_1, and the probability that the adversary can distinguish a random string from $n||s||x_1||\cdots||x_s||y_1||\cdots||y_t$, given the ciphertexts (i.e., break semantic security), is denoted by p_2. In SecureEdge, an adversary can win the soundness game without storing the challenged file with probability at most $\epsilon = \frac{p_1 p_2}{p}$.

Theorem 2. If the signature is secure, no adversary can defeat the soundness of publicly verifiable SecureEdge with non-negligible probability, that is, TPA will not accept data integrity proofs generated by an adversary without maintaining the challenged file correctly.

We can prove the theorem with a series of games.

Game 0. The first game, Game 0, is simply the challenge game defined in section 4.2.2, with the changes for public verifiability, in which the verifier is the data user or the TPA.

Game 1. Game 1 is the same as Game 0, with one difference. The challenger keeps a list of all the signatures returned in TagGen queries. If the adversary generates a signature $SSig_{ssk}(\cdot)$ in TagGen or VeriP, that 1) the signature is valid under ssk, but 2) is not a signature generated by the challenger in TagGen queries, the challenger declares failure and aborts.

Obviously, the difference between Game 0 and Game 1 is the adversary's success probability to construct a signature $SSig_{ssk}(\cdot)$ under the private key ssk. If the signature $SSig_{ssk}(\cdot)$ is existential unforgeable under the mathematical hard problem, the gap between Game 0 and game 1 is negligible.

Game 2. Game 2 is the same as Game 1, with one difference. The challenger keeps a list of its responses to TagGen queries and ReplicaGen queries made by the adversary. The challenger could observe the tag generation and challenge-proof process made by the adversary. If all these instances are successful, but the adversary's aggregated tag σ^u is not equal to $\sum_{(i,\nu_i)\in Q} \nu_i \sigma_i$, the challenger declares failure and aborts.

Suppose the file that leads to abort has n blocks and the file identity is F_{ID}, which is stored on the edge server E_u and generates the aggregated blocks μ_j^u, containing the randomized blocks m_i^u with r_j^u for $1 \le j \le s$. Suppose $Q = \{(i, \nu_i)\}$, $i \in I$, is the query that causes the challenger to abort, and that the adversary's response to that query is $\{\mu_1^{'u}, \cdots, \mu_s^{'u}\}$ and the aggregated tag $\sigma^{'u}$. Additionally, suppose the expected response is $\{\mu_1^u, \cdots, \mu_s^u\}$ and the aggregated tag σ^u. According to the correctness of SecureEdge, the expected response satisfies the verification equation (4.7):

$$\hat{e}(g^{\sigma^u}, g) \cdot \prod_{j=1}^{s} \hat{e}(S_j^{\sum\limits_{(i,\nu_i)\in Q} \nu_i}, R_j^u) = \hat{e}(S^{\sum\limits_{(i,\nu_i)\in Q} \nu_i H(F_{ID}||i)}, g) \cdot \hat{e}(\prod_{j=1}^{s} S_j^{\mu_j^u}, g). \tag{4.12}$$

Because the challenger aborts, we know that $\sigma^{'u} \neq \sigma^u$, but $\sigma^{'u}$ passes the verification, so we have

$$\hat{e}(g^{\sigma^{'u}}, g) \cdot \prod_{j=1}^{s} \hat{e}(S_j^{\sum_{(i,\nu_i)\in Q} \nu_i}, R_j^u) = \hat{e}(S^{\sum_{(i,\nu_i)\in Q} \nu_i H(F_{ID}||i)}, g) \cdot \hat{e}(\prod_{j=1}^{s} S_j^{\mu_j^{'u}}, g). \tag{4.13}$$

If $\mu_j^u = \mu_j^{'u}$ for each j, it follows from the verification equation that $\sigma^u = \sigma^{'u}$, which contradicts our assumption above. Thus, we define $\Delta\mu_j^u = \mu_j^{'u} - \mu_j^u$, for $1 \leq j \leq s$, so that at least one of $\{\Delta\mu_j^u\}$ is nonzero.

We divide the verification equation for the forged aggregated tag $\sigma^{'u}$ by the verification equation for the expected tag σ^u, we obtain

$$\hat{e}(g^{\sigma^{'u} - \sigma^u}, g) = \hat{e}(\prod_{j=1}^{s} S_j^{\mu_j^{'u} - \mu_j^u}, g). \tag{4.14}$$

Then, we yield

$$\hat{e}(g^{\sigma^{'u} - \sigma^u}, g) = \hat{e}(g^{\sum_{j=1}^{s} x_j \Delta\mu_j^u}, g). \tag{4.15}$$

Therefore, we have $\sigma^{'u} - \sigma^u = \sum_{j=1}^{s} x_j \Delta\mu_j^u$. With this linear property, the adversary can obtain all x_j, for $1 \leq j \leq s$.

Thus, if there is a non-negligible difference between the adversary's probabilities of success in Games 1 and Game 2, the values of x_j, for $1 \leq j \leq s$, can be obtained, which contradicts our assumption in SecureEdge. Without the knowledge of $\{x_1, \cdots, x_s\}$, the probability that the remaining part equal is $1/p$, which is the negligible because p is a large prime. Therefore, the gap between Game 1 and Game 2 is negligible.

Suppose the probability that an adversary can forge a digital signature is denoted by p_1. In SecureEdge, an adversary can win the soundness game without storing the challenged file with probability at most $\epsilon = \frac{p_1}{p}$.

Theorem 3. SecureEdge ensures privacy against TPA, guaranteeing that no adversary, including the TPA, can learn any information about the data owner's edge data during the integrity verification process.

The view of the curious TPA, $View_{TPA^*}(EID_u, pk, chal, M^1, \cdots, M^t, F_{id}, tags, P^1, \cdots, P^t)$, where $P^u = \{EID_u, \mu_1^u, \cdots, \mu_s^u, \sigma^u\}$, $M^1, \cdots, M^t$ are the multi-replicas, for $1 \leq u \leq t$ is computationally indistinguishable from the simulated view $\mathcal{S}(F_{id}, pk, chal, M_*^1, \cdots, M_*^t, tags_*, P_*^1, \cdots, P_*^t)$, where $M_*^1, \cdots, M_*^t, tags_*, P_*^1, \cdots, P_*^t$ are random values. Therefore, the privacy of the edge data is preserved against the verifier in SecureEdge, including the TPA.

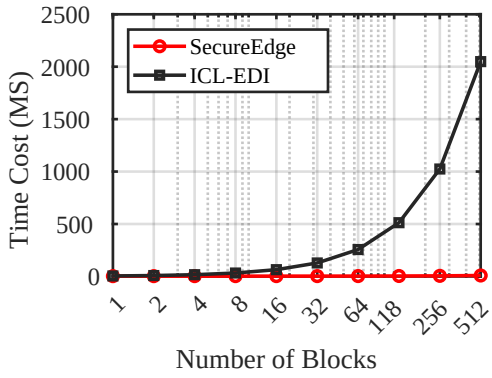
4.5 Performance Evaluation

In this section, we evaluate the performance of SecureEdge in terms of computational, communication, and storage overhead.

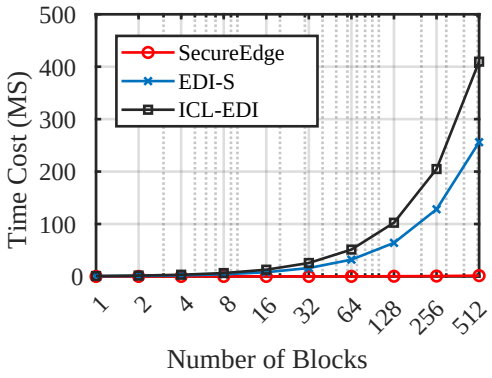
SecureEdge is designed with lightweight cryptographic algorithms that avoid costly operations such as elliptic curve scalar multiplication and bilinear pairings on the data owner side. Instead, it employs efficient symmetric-key primitives, including symmetric encryption, MACs, PRPs, and PRFs, which can be instantiated using well-established constructions like AES and HMAC [125]. Extensive experiments were conducted to evaluate the system's performance on both end-user devices and edge servers. All benchmarks were performed on a laptop equipped with a 12th Gen Intel Core i7-1280P CPU and 32 GB of RAM. The cryptographic parameters include a 256-bit prime modulus p, SHA-256 as the hash function, HMAC-SHA256 as the MAC algorithm, and AES for symmetric encryption. To implement number-theoretic based methods of cryptography, we utilized the JPBC library. For achieving bilinear pairing, we employed Type-A pairing. The implementation is under the elliptic curve. The elliptic curve is represented as $y^2 = x^3 + x$ and defined over F_p, with an embedding degree of $k = 6$. For the Type-A pairing, we set the prime subgroup order p to 256 bits and the base-field prime q to 1536 bits. The pairing parameters are generated using the JPBC configuration `TypeACurveGenerator(256,1536)`. In the resulting JPBC parameter file, the subgroup order is denoted by r, which corresponds to p here, while q denotes the base-field prime.

SecureEdge achieves high efficiency across all core operations, including tag generation, proof generation, and proof verification. For instance, when a data owner processes a 1 MB file divided into 512 blocks, each further split into 64 sectors for 4 edge servers, the TagGen algorithm completes in approximately 18 milliseconds. During data integrity verification in which 32 blocks are challenged, the edge server takes under 7 milliseconds to produce a proof using the Proof algorithm, while the data owner requires only 3.2 milliseconds to verify it using the VeriP algorithm. For the public verification, the time cost is larger than the cost in private verification. It take about 21 milliseconds to verify the proof.

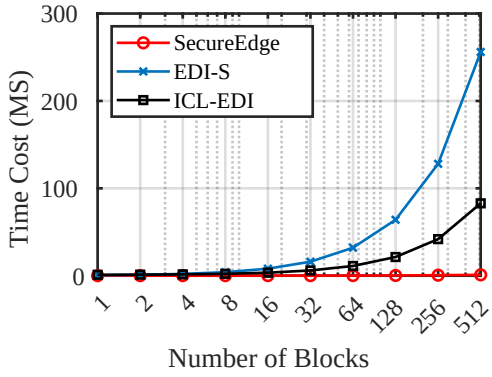
We also compare the computational efficiency of SecureEdge with existing schemes, including EDI-S [52] and ICL-EDI [53]. The results are presented in Fig. 4.6. Since EDI-S does not support end-user tag generation, its performance is excluded in Figs. 4.6(a) and 4.6(d). Figs. 4.6(a), 4.6(b), and 4.6(c) show the time costs of the core operations, tag generation, proof generation, and proof verification, with respect to the number of blocks, assuming there is a single edge server, 32 sectors per block, and half of the blocks are challenged during integrity verification. Meanwhile, Figs. 4.6(d), 4.6(e), and 4.6(f) present



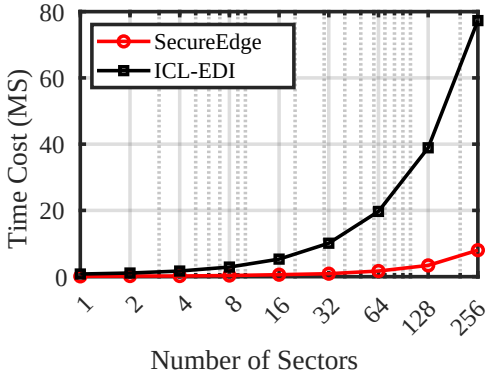
(a) Time Cost of Data Owners in TagGen



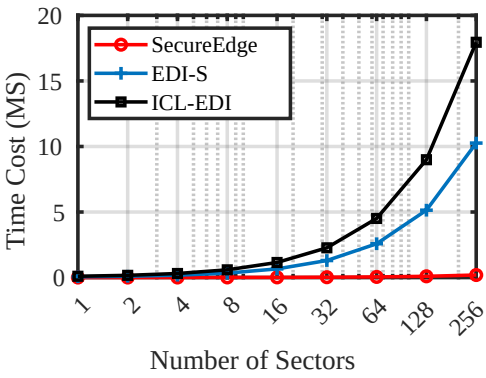
(b) Time Cost of Edge Servers in Proof



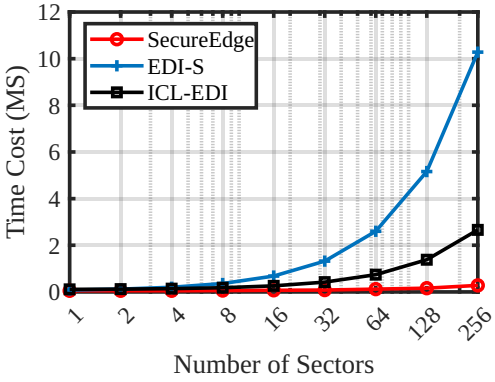
(c) Time Cost of Data Owners in VeriP



(d) Time Cost of Data Owners in TagGen



(e) Time Cost of Edge Servers in Proof



(f) Time Cost of Data Owners in VeriP

Figure 4.6: Computational Overhead Comparison of SecureEdge and Existing Schemes

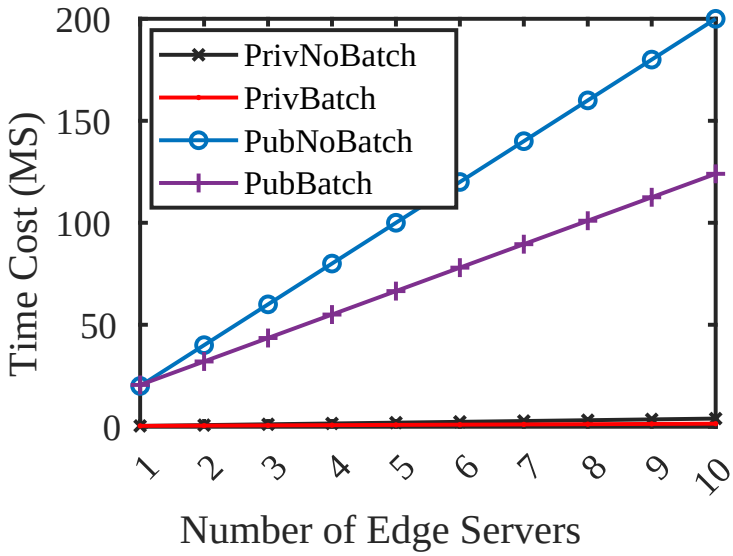


Figure 4.7: Comparison between Private and Public Verification and Batch Verification

the time costs of tag generation, proof generation, and proof verification with respect to the number of sectors, assuming there is one edge server and a single block. Fig. 4.6 demonstrates that SecureEdge significantly outperforms both EDI-S and ICL-EDI. This efficiency gain is attributed to SecureEdge's use of lightweight symmetric-key primitives, in contrast to the reliance of EDI-S and ICL-EDI on computationally intensive elliptic curve scalar multiplications for generating authentication tags and data integrity proofs.

In addition, we compare the computational performance of SecureEdge under public and private verification modes, and demonstrate the time savings achieved through batch verification. In Fig. 4.7, the number of blocks is 512, the number of sectors is 64, and half of the number of blocks are challenged. It shows that in both private and public verification settings, batch verification of proofs significantly reduces the verifier's computational overhead. Moreover, private verification is notably more efficient than public verification. This efficiency is particularly beneficial to data owners, while public verification enables the TPA to perform integrity checks, ensuring the transparency and auditability of the verification process.

SecureEdge does not impose significant overhead on communication. In *TagGen*, the data owner uploads the file to the edge servers. In addition to the file, the extra communication overhead brought by SecureEdge is $1792 + 256n + 768t + 1280s$, where n is the number of blocks, s is the number of sectors, and t is the number of edge servers. The size of the challenge message in *Chal* is $1280 + 512|Q|$, where $|Q|$ is the number of chal-

lenged blocks. In *Proof*, an edge server generates the proof in response to the challenge, the message size of which is $768 + 256s$. In *VeriP*, the verifier retrieves τ_0 and τ_1, whose message sizes are $768 + 256t + 256s$ and $1024 + 512t + 1024s$, respectively. In terms of storage cost, the data owners and the data users do not need to maintain any data. All data is uploaded to the edge server. Each edge server maintains the file and additional $1792 + 256n + 768t + 1280s$ bit data.

4.6 Summary

In this chapter, we have proposed SecureEdge, an efficient and secure multi-replica data integrity verification framework for MEC, built upon lightweight cryptographic primitives. SecureEdge supports both private and public verifiability, enabling data owners and TPAs to efficiently verify the integrity of their data stored on distributed edge servers. To balance security and efficiency, SecureEdge achieves two key properties: high computational efficiency and secure multi-replica storage. Specifically, authentication tag generation and proof verification incur minimal computational overhead for data owners, while secure multi-replica storage is ensured by enforcing replica diversity through randomization, effectively preventing collusion attacks among edge servers. Moreover, SecureEdge supports data dynamics and privacy preservation, ensuring that data updates can be verified securely and that sensitive user information remains protected during public auditing. These properties make SecureEdge particularly suitable for ensuring data integrity in MEC scenarios.

Chapter 5

SVTAgg: Secure, Verifiable, and Traceable Federated Learning for Mobile Edge Computing[2]

5.1 Introduction

Edge intelligence [127] leverages edge computing [128] to deploy AI services closer to data sources, offering a scalable solution to the distributed learning demands of massive IoT ecosystems and the growing need for real-time data analytics, decision-making, and productivity enhancement. As a key enabler of edge intelligence, FL enables collaborative model training directly at end devices without exposing raw data, thereby preserving privacy while supporting distributed AI workloads [129]. Building on this paradigm, EFL [130] integrates FL with multi-tier edge computing, allowing decentralized devices, such as IoT sensors and mobile phones, to train local models and offload intermediate aggregation to nearby edge servers. These edge servers act as intermediary coordinators that efficiently aggregate local updates, mitigate device heterogeneity, reduce the burden on cloud servers, and significantly improve system responsiveness. EFL offers several advantages, including reduced latency and bandwidth overhead, accelerated model training at both device and edge levels, and enhanced support for latency-sensitive applications such as anomaly detection in IoT networks and autonomous driving systems [131].

[2]A version of this chapter has been published in IEEE Transactions on Dependable and Secure Computing [126].

While EFL preserves data locality by allowing each end device to retain raw data and upload only their local model updates to the edge servers, these updates themselves may expose sensitive information. Despite not containing explicit data samples, gradient vectors and model parameters are known to be vulnerable to inference and model inversion attacks [132], enabling an adversarial observer to reconstruct features or even approximate original inputs. Although local model updates are usually encrypted during model uploading to protect against external eavesdroppers, a curious edge server, who performs intermediate aggregation, can still exploit the uploaded gradients to infer private data characteristics. This concern is amplified in well-generalized deep models such as DenseNet and ResNet [133], where the expressive gradient space has been shown to leak substantial information about the underlying training samples. Consequently, without dedicated protection mechanisms, EFL remains susceptible to serious privacy leakage despite its decentralized design.

Recently, secure model aggregation protocols [134] have been proposed in EFL to protect the privacy of local model updates uploaded from end devices to the edge servers. By encrypting or masking local gradients, these protocols enable edge servers to perform intermediate aggregation without learning any individual participant update, thereby mitigating inference and inversion attacks from curious edge nodes. Although secure aggregation effectively prevents edge servers from extracting sensitive information, it does not ensure that an edge server has executed the aggregation correctly [135]. A dishonest or lazy edge server may skip certain updates, manipulate intermediate results, or return arbitrary aggregated outputs, and such misbehavior would propagate to the cloud-level aggregation without being detected. This integrity issue is particularly significant in EFL, where edge servers play a critical coordination role in model aggregation and their outputs directly influence the global model's convergence and performance. Although prior work has proposed privacy-preserving and verifiable aggregation mechanisms for traditional FL [93, 82], their application to edge-assisted hierarchical aggregation at edge servers remains largely unexplored. Moreover, verifiable secure aggregation introduces additional verification-related information at the edge server. The potential leakage of local gradients through such verifiable tags has not been thoroughly investigated. Consequently, a crucial gap remains in ensuring both privacy and verifiable integrity for model aggregation in EFL systems.

In addition to privacy preservation and aggregation correctness, EFL introduces critical concerns related to client traceability and accountability. In EFL, the cloud server lacks direct visibility into individual updates submitted by end devices, making it difficult to accurately assess client contributions. This limited observability complicates contribution auditing, incentive allocation, and the attribution of malicious behavior. The absence of reliable traceability becomes especially problematic in adversarial or incentive-driven envi-

ronments. Malicious clients may upload poisoned or low-quality updates to degrade global model performance, while free-riding participants may attempt to claim rewards without performing meaningful local training. Although a few secure aggregation protocols integrate digital signatures to authenticate client identities or contributions, these approaches often leak information about local model gradients, incur additional communication overhead, and even create technical dilemmas in EFL settings. Individual signatures become unverifiable for the cloud server after intermediate aggregation at the edge without compromising privacy. Therefore, designing effective traceability mechanisms in EFL requires carefully balancing gradient privacy, aggregation correctness, and verifiable contribution attribution.

In this chapter, we propose SVTAgg [126], a secure, verifiable, and traceable aggregation protocol tailored for EFL. SVTAgg builds upon the state-of-the-art Secure Aggregation (SecureAgg) protocol by Bonawitz et al. [26], which employs a pairwise double-masking mechanism based on secret sharing to protect the confidentiality of local model gradients. To extend SecureAgg to hierarchical EFL settings, we integrate it with homomorphic signatures and proxy re-signatures, enabling verifiable aggregation across both edge servers and the central cloud server. This design supports scalable and efficient global model construction while preserving gradient privacy in multi-tier architectures. A key benefit of incorporating homomorphic signatures and proxy re-signatures is the ability to cryptographically verify the correctness of aggregated gradients, thereby mitigating potential edge-server misbehavior such as selective update dropping, aggregation manipulation, or arbitrary result fabrication. Furthermore, the produced homomorphic verifiable tags provide a mechanism for client traceability, allowing the system to authenticate and identify participating end devices when accountability or dispute resolution is required. However, enabling public verifiability introduces new privacy risks. In particular, verifiable tags may become vulnerable to offline guessing attacks, where an adversary attempts to infer local model gradients by exhaustively testing candidate parameters against publicly verifiable tags. To prevent such gradient leakage, SVTAgg carefully integrates the masking mechanism in SecureAgg to disable public verifiability at the individual update level. Instead, only verifiable tags corresponding to aggregated model updates are verifiable. This design preserves aggregation-level correctness verification and traceability while preventing adversaries from recovering individual local gradients.

The main contributions of this chapter can be summarized as follows:

- We propose SVTAgg, the first local model aggregation protocol that simultaneously achieves gradient privacy, aggregation verifiability, and client traceability.

- By integrating homomorphic signatures and proxy re-signatures, SVTAgg enables secure and verifiable hierarchical aggregation across both edge servers and the cloud, supporting scalable global model construction in multi-tier EFL architectures.
- SVTAgg guarantees the correctness of aggregation performed by potentially untrusted edge servers and supports the traceability of participating end devices, ensuring accountability without compromising the privacy guarantees of EFL.
- SVTAgg prevents offline guessing attacks on verifiable tags by utilizing the masking mechanism in SecureAgg, ensuring that individual local gradients remain confidential while preserving verifiability at the aggregate level.

5.2 Models and Design Goals

In this section, we first introduce the main architecture of EFL and then describe the threat model and the design goals.

5.2.1 System Model

The EFL system adopts a hierarchical architecture consisting of three layers: clients, edge servers, and a cloud server. Clients are end devices, such as smart vehicles, unmanned aerial vehicles, and portable medical terminals, that collect data about users and their surrounding environments and perform local model training, optionally assisted by AI training toolkits. The edge servers comprise local servers and edge devices in mobile networks, such as roadside units, micro base stations, and macro base stations, which are deployed in close proximity to the clients. The cloud server A centrally coordinates the training of neural network models with a set of edge servers $\{A_j\}_{j=1}^{L}$. Each edge server A_j further coordinates local training with a randomly selected subset of clients within its coverage area. The clients associated with A_j are denoted by $U_j = \{u_{i,j}\}_{i=1}^{N_j}$, where N_j represents the number of clients. Collectively, all clients $\{U_j\}_{j=1}^{L}$ collaborate to train a shared global neural network model Ω under the EFL paradigm.

The learning objective of EFL can be formulated as the following optimization problem:

$$\min_{\boldsymbol{w}\in\mathbb{W}^m} \frac{1}{N}\sum_{j=1}^{L}\sum_{i=1}^{N_j}\mathbb{E}_{\xi_{i,j}\sim D_{i,j}}\left[l(\boldsymbol{w},\xi_{i,j})\right], \tag{5.1}$$

where $N = \sum_{j=1}^{L} N_j$ is the total number of clients, $l(\cdot)$ denotes the local loss function over training samples $\xi_{i,j}$ drawn from client $u_{i,j}$'s local dataset $D_{i,j}$, $\boldsymbol{w}$ represents the global model parameters, and $\mathbb{E}[\cdot]$ denotes expectation.

An EFL system follows the pipeline described below.

- FL Task Release: Upon receiving a FL task, the cloud server A initializes the global model $\boldsymbol{w}_a^{(0)}$.
- Global Model Download: During training round r, each edge server A_j downloads the current global model $\boldsymbol{w}_a^{(r)}$ from the cloud server A and distributes it to selected clients $u_{i,j}$ within its coverage area.
- Local Model Training: Each client $u_{i,j}$ performs local training using Stochastic Gradient Descent (SGD) and updates its local model as

$$\boldsymbol{w}_{i,j}^{(r+1)} = \boldsymbol{w}_a^{(r)} - \eta \nabla l\big(\boldsymbol{w}_a^{(r)}, D_{i,j}\big), \tag{5.2}$$

  where η denotes the local learning rate.
- Edge Aggregation: The updated local models are periodically transmitted to the nearby edge server A_j, which aggregates the received updates from clients in U_j.
- Cloud Aggregation: Each edge server A_j forwards its aggregated model update $\boldsymbol{w}_j^{(r+1)}$ to the cloud server A. The cloud server then aggregates all edge-level results to obtain the updated global model $\boldsymbol{w}_a^{(r+1)}$.
- Global Model Update: The cloud server A disseminates $\boldsymbol{w}_a^{(r+1)}$ to all edge servers $\{A_j\}_{j=1}^{L}$, which subsequently forward it to the selected clients. Clients update their local models accordingly and proceed to the next training round.

This iterative process continues until a stopping criterion is met, typically when the global model converges or reaches a target accuracy. The system architecture of SVTAgg is shown in Fig. 5.1.

5.2.2 Threat Model

The sensitive information includes each client's training data samples, sample labels, and local model updates in every training round of EFL. The clients are assumed to be semi-honest (honest-but-curious): They follow the prescribed protocol correctly but may attempt to infer private information about other targeted clients, such as the distribution of

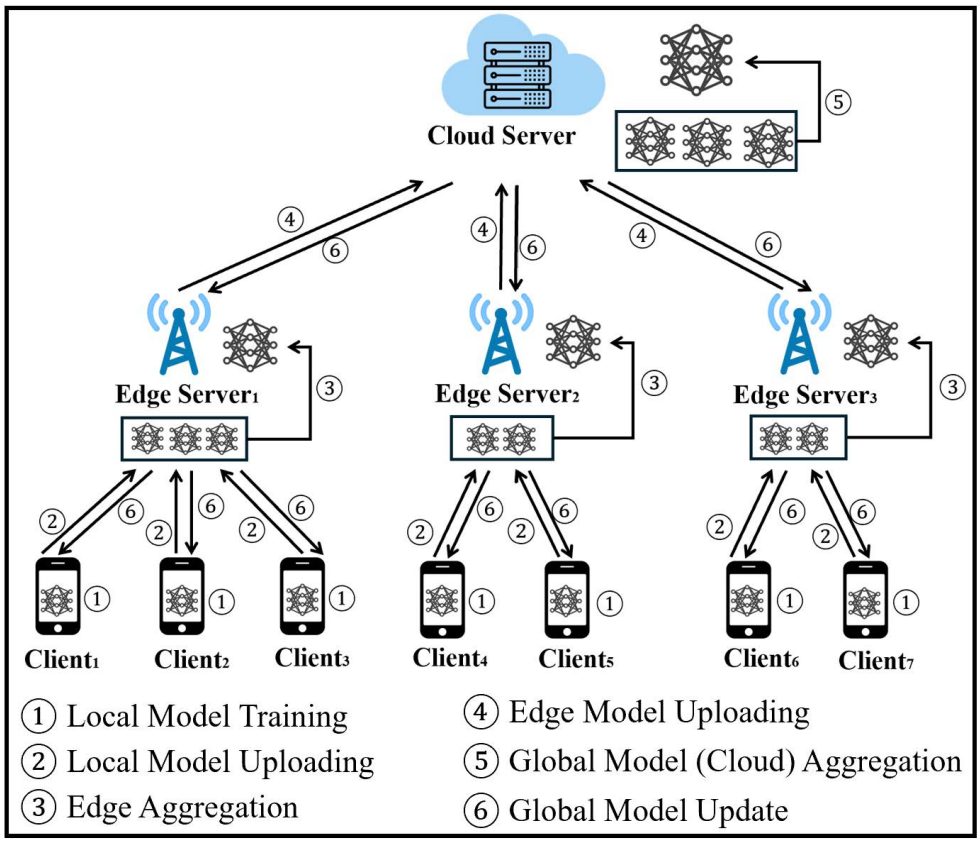


Figure 5.1: System Architecture of SVTAgg

their training data or even their local models based on messages observed from communication channels, without engaging in active protocol deviations.

The cloud server is assumed to be fully trusted and is responsible for initiating EFL tasks for training target models. It coordinates all participating edge servers and oversees the overall model training process. In contrast, edge servers are not fully trusted by the cloud server, as any edge server may join the EFL task dynamically. We do not assume that all edge servers are malicious; however, they are considered potentially untrusted and may arbitrarily deviate from the prescribed protocol during the coordination and aggregation of client updates. In particular, a malicious edge server may observe all received messages, selectively omit or manipulate client updates, inject fabricated aggregation results, or attempt to infer private information from protocol transcripts across multiple training rounds. We assume that the adversary has full control over corrupted parties but cannot

break standard cryptographic primitives (e.g., commitment binding, MAC unforgeability, or zero-knowledge proof soundness) or compromise the trusted execution environments of honest clients.

Finally, due to the inherent mobility of client devices, an arbitrary fraction of semi-honest clients may drop out during protocol execution in EFL. This dynamic environment highlights the importance of designing mechanisms that tolerate client dropouts while still achieving client traceability and accountability in each round of model aggregation, without compromising security or correctness guarantees.

5.2.3 Design Goals

We aim to protect the clients' model updates, prevent the misbehavior of the edge servers, and trace the clients who contribute to the final global model in EFL.

- **Security.** SVTAgg preserves the confidentiality of each client's local model update against semi-honest and malicious entities, including the untrusted edge servers and curious clients. No adversary can learn any information about an individual update beyond what is revealed by the aggregated model.
- **Verifiability.** SVTAgg enables the cloud server to verify the correctness of edge-level aggregations. Any deviation from the prescribed aggregation protocol by a malicious edge server is detected by the cloud server.
- **Traceability.** SVTAgg supports traceability, allowing the identification of clients who contributed to a particular local model. The contribution of each client on the local model training can be recognized.

5.3 SVTAgg

In this section, we present the technical overview and present the detailed constructions of SVTAgg in EFL.

5.3.1 Overview of SVTAgg

EFL adopts a hierarchical architecture in which local model updates are aggregated at both the edge and cloud levels. To achieve the design goals of security, verifiability, and

traceability, it is necessary to extend the existing EFL framework with appropriate cryptographic primitives. To protect local model gradients contributed by clients, we leverage pairwise double masking to protect local model parameters before submitting them to the edge server. This technique is both effective and efficient, and naturally supports client dropouts, making it well suited for EFL environments.

Our main novelty lies in extending pairwise double masking to additionally support verification and traceability. To this end, we employ homomorphic signatures, rather than the homomorphic hash commonly used in the literature, which typically provide verifiability only and potentially need extra digital signatures to prevent forgery attacks. The homomorphic verifiable tags generated by homomorphic signatures can be aggregated at the edge servers, enabling verification of aggregated results as well as identification of contributing clients at the cloud server. Although conceptually straightforward, integrating homomorphic signatures into EFL introduces several nontrivial challenges.

First, homomorphic signatures generated by different clients are signed under different keys and over different model parameters, which prevents direct aggregation at the edge level. To address this issue, we integrate homomorphic signatures with proxy re-signatures, allowing an edge server to re-sign local model updates from multiple clients to a common signer, namely the cloud server. As a result, the aggregated re-signatures produced by the edge server become valid verifiable tags on the aggregated local models.

Second, homomorphic signatures are generally publicly verifiable, which improves transparency of verification results but also introduces a security vulnerability. Specifically, a curious edge server may launch offline guessing attacks by exhaustively searching for local model parameters consistent with the publicly verifiable tags, exploiting the relatively small parameter space of model updates. To mitigate this risk, we draw inspiration from designated verifier signatures, where only a specific verifier can validate a signature. However, directly designating the cloud server as the sole verifier would undermine transparency. In SVTAgg, we instead combine this idea with pairwise double masking: homomorphic signatures become verifiable only after re-signing and aggregation, when the random masks added by clients are removed. If a client drops out, the corresponding secret keys reconstructed via secret sharing are required to complete verification, which is consistent with the recovery mechanism used in pairwise double masking to ensure correct edge-level aggregation.

Third, it is critical to ensure that the same local model gradients are used both to generate the masked updates and to produce the corresponding homomorphic verifiable tags. While this requirement is not primarily intended to prevent malicious client behavior, inconsistencies may arise due to client-side operational errors or other unintended fac-

tors. Such inconsistencies can lead to verification failures of edge-level aggregation results, thereby undermining the correctness guarantees of the protocol. To prevent verification errors caused by mismatched gradients, it is essential to enable consistency verification without revealing the underlying model parameters to edge servers. To address this challenge, we incorporate zero-knowledge proofs, allowing each client to prove that its masked update and corresponding homomorphic signatures are derived from the same set of model gradients, without disclosing any sensitive information.

5.3.2 Detailed SVTAgg

We present the detailed procedures of SVTAgg. It is bootstrapped by a system initialization phase executed by the cloud server. The cloud server initializes the system parameters based on a security parameter κ. Specifically, given the security parameter κ, let p be a large prime and generate a bilinear map $\hat{e} : G \times G \to G_T$, here G and G_T are cyclic groups of the prime order p, g, g_0, g_1 are generators of G, hash functions $H : \{0,1\}^* \to G$ and $H_{\mathsf{FS}} : \{0,1\}^* \to Z_p^*$, and a pseudo-random generator PRG. The system parameters include $(p, \hat{e}, G, G_T, g, g_0, g_1, H, H_{\mathsf{FS}}, PRG)$.

SVTAgg operates over multiple training rounds and involves coordinated interactions among the cloud server, edge servers, and clients' end devices. The protocol is conceptually divided into two phases: key preparation and model training.

During the key preparation phase, all entities initialize the required cryptographic materials. Specifically, the cloud server, edge servers, and clients generate their respective public–private key pairs, and edge servers additionally obtain proxy re-signing keys from the cloud server. Meanwhile, clients securely generate the values needed to protect local model gradients during training, including mask seeds used for pairwise masking of local models, and secret shares of clients' private keys and random mask seeds for model unmasking and aggregation. The detailed procedures of the key preparation phase are illustrated below. $s \xleftarrow{\$} G$ denotes s is randomly chosen from a group G. The notations of SVTAgg are presented in Table 5.1.

- **Round 1 (GenerateKey)**

  Client $u_{i,j}$:

  – Generate the private-public key pair $(SK_{i,j}, PK_{i,j})$, $SK_{i,j} = (\alpha_{i,j}, \beta_{i,j})$, and $PK_{i,j} = (g^{\alpha_{i,j}}, g^{\beta_{i,j}})$, where $\alpha_{i,j} \xleftarrow{\$} Z_p^*$, $\beta_{i,j} \xleftarrow{\$} Z_p^*$ and send the public key $PK_{i,j}$ to the edge server A_j.

Table 5.1: Notations of SVTAgg

Symbol	Description
A	Cloud server
A_j	Edge server j
$u_{i,j}$	Client i under edge server A_j
U_j	Set of clients under edge server A_j
N_j	Number of clients under A_j
$x_{i,j}$	Local model update (gradients) of client $u_{i,j}$
$X_{i,j}$	Masked model update of client $u_{i,j}$
z_j	Aggregated model update at edge server A_j
Z	Global aggregated model at the cloud server
$\gamma_{i,j}$	Random seed for secondary masking
$\sigma_{i,j}$	Homomorphic signature of the local model $x_{i,j}$ for client $u_{i,j}$
σ_j	Aggregated verifiable tag at edge server A_j
$PK_{i,j}$, $SK_{i,j}$	Public and private keys of client $u_{i,j}$
$RK_{i,j}$	Proxy re-signing key for client $u_{i,j}$
$H(\cdot)$	Cryptographic hash function
$PRG(\cdot)$	Pseudorandom generator
$S.Share$, $S.Recon$	Secret sharing and reconstruction functions
$\pi_{i,j}$	Zero-knowledge proof generated by client $u_{i,j}$

Edge Server A_j:

- Generate the private-public key pair (s_j, g^{s_j}), where $s_j \xleftarrow{\$} Z_p^*$.
- Receive the public keys from at least t_j clients (denote with $N_{j,0}$ this set of clients, suppose $|N_{j,0}| > t_j$), where t_j is the threshold of the Shamir's t_j out of $N_{j,0}$ protocol used for the edge server A_j.
- Forward the received public keys to all clients $u_{i,j} \in N_{j,0}$ and the cloud server.

Cloud Server A:

- Generate the private-public key pair (b, g^b), where $b \xleftarrow{\$} Z_p^*$.
- For each participating client $u_{i,j}$ in the coverage area of the edge server A_j, compute the proxy re-signing key for $u_{i,j}$ as $RK_{i,j} = g^{\beta_{i,j}b}$ and send $RK_{i,j}$ to the edge server A_j.

- **Round 2 (ShareKeys)**

Client $u_{i,j}$:

- Receive the message from the edge server A_j,
- Select a random value $\gamma_{i,j} \xleftarrow{\$} Z_p^*$ as a random mask seed for the secondary masking of its local model gradients and generate the shares of $\gamma_{i,j}$ as $\{m, \gamma_{im,j}\}_{m \in N_{j,0}} = S.Share(\gamma_{i,j}, t_j, |N_{j,0}|)$, where $\gamma_{im,j}$ is the share of the client $u_{i,j}$ to the client $u_{m,j}$.
- Generate the shares of $\alpha_{i,j} \xleftarrow{\$} Z_p^*$ as $\{(m, \alpha_{im,j})\}_{m \in N_{j,0}} = S.Share(\alpha_{i,j}, t_j, |N_{j,0}|)$, where $\alpha_{im,j}$ is the share of the private key of the client $u_{i,j}$ to the device $u_{m,j}$.
- Compute $C_{im,j} = Enc(KA.agree(g^{\alpha_{i,j}}, g^{\alpha_{m,j}}), i||m||\gamma_{im,j}||\alpha_{im,j}), m \in N_{j,0} \backslash \{i\}$, where $Enc()$ is a symmetric encryption with the secret key $KA.agree(g^{\alpha_{i,j}}, g^{\alpha_{m,j}})$. $r_{im,j} = KA.agree(g^{\alpha_{i,j}}, g^{\alpha_{m,j}})$ is a mask seed used for pairwise masking of the client $u_{i,j}$
- Send $C_{im,j}$ to the edge server A_j.

Edge Server A_j:

- Receive the messages from at least t_j clients (denote with $N_{j,1}$ this set of clients, $N_{j,1} \subseteq N_{j,0}$) who generate the secret shares. If the size of $N_{j,1}$ is less than t_j, abort.
- Broadcast $\{C_{im,j}\}_{i \in N_{j,1}}$ to each client $u_{m,j}, m \in N_{j,1} \backslash \{i\}$.

- **Round 3 (ObtainShares)**

  Client $u_{i,j}$:

  - Receive $\{C_{mi,j}\}_{m \in N_{j,1} \backslash \{i\}}$ from the edge server A_j.
  - Calculate the shared key with every client $m \in N_{j,1} \backslash \{i\}$ as $r_{mi,j} = KA.agree(g^{\alpha_{m,j}}, g^{\alpha_{i,j}})$ and decrypt $\{C_{mi,j}\}_{m \in N_{j,1} \backslash \{i\}}$ with $r_{mi,j}$ to obtain $m||i||\gamma_{mi,j}||\alpha_{mi,j}$ for each $m \in N_{j,1} \backslash \{i\}$.

In the model training phase, clients' end devices locally train their models and protect the resulting model gradients using pairwise double masking, producing encrypted model updates (model ciphertexts). In parallel, clients generate homomorphic verifiable tags using homomorphic signatures. To ensure consistency between the masked model updates and the corresponding verifiable tags, each client also generates a zero-knowledge proof, attesting that both are derived from the same local model gradients without revealing the gradients themselves. The model ciphertexts, homomorphic verifiable tags, and zero-knowledge proof transcripts are then submitted to the corresponding edge server. Upon

receiving these information, the edge server verifies the zero-knowledge proofs, performs edge-level aggregation to obtain aggregated local model parameters, and executes proxy re-signing on the homomorphic verifiable tags using the proxy re-signing key issued for the cloud server, and aggregate the re-signed verifiable tags. This re-signing step ensures that the aggregated verifiable tags can be validated at the cloud level. During edge-level aggregation, clients may be involved in providing their secret shares to reconstruct the private keys of dropped-out clients, as well as the random mask seeds of surviving clients. They are used to mask local model gradients. Subsequently, the cloud server conducts cloud-level aggregation to derive the global model, verifies its correctness using the re-signed homomorphic verifiable tags, and distributes the updated global model to clients via the edge servers for the next training round. The detailed steps of the model training phase are shown below.

- **Round 1 (LocalMask)**

  Client $u_{i,j}$:

  - Train a model with the local data to generate its local model gradients $x_{i,j}$.
  - Calculate a mask with the mask seed $r_{im,j}, m \in N_{j,1} \backslash \{i\}$ for pairwise masking of the local gradients as $r_{i,j} = \sum_{i<m} PRG(r_{im,j}) - \sum_{m<i} PRG(r_{mi,j})$ and encrypt the local gradients as $X_{i,j} = x_{i,j} + PRG(\gamma_{i,j}) + r_{i,j}$.
  - Calculate the homomorphic signature $\sigma_{i,j} = (H(PK_{i,j}||j||g^b) g_0^{x_{i,j}} g_1^{r_{i,j}})^{\frac{1}{\beta_{i,j}}}$.
  - Calculate the zero-knowledge proof:

$$PoK_{i,j} = \left\{ \begin{array}{r} (x_{i,j}, PRG(\gamma_{i,j}), r_{i,j}, \beta_{i,j}) : X_{i,j} = x_{i,j} + PRG(\gamma_{i,j}) + r_{i,j} \wedge \\ Y_{i,j} = g_0^{X_{i,j}} = g_0^{x_{i,j}+PRG(\gamma_{i,j})+r_{i,j}} \wedge \\ \sigma_{i,j} = (H(PK_{i,j}|j||g^b) g_0^{x_{i,j}} g_1^{r_{i,j}})^{\frac{1}{\beta_{i,j}}} \end{array} \right\}.$$

  - Send the message $\Omega_i = (X_{i,j}, \sigma_{i,j}, PoK_{i,j})$ to the edge server A_j.

  Edge Server A_j:

  - Receive the messages from at least t_j clients (denote with $N_{j,2}$ this set of clients); Otherwise, aborts.
  - Send the list of $N_{j,2}$ to the clients in $N_{j,2}$.

- **Round 2 (Aggregate&Unmask)**

  Client $u_{i,j}$:

  - Receive from the edge server A_j a list of clients in $N_{j,2}$, verify that $N_{j,2} \subseteq N_{j,1}$ and $|N_{j,2}| > t_j$; Otherwise, abort.
  - Send the secret shares of private keys of dropout clients and random mask seeds of surviving clients to the edge server A_j, which consists of $\gamma_{mi,j}$ for clients in $N_{j,2} \setminus \{i\}$ and $\alpha_{mi,j}$ for clients in $N_{j,1} \setminus N_{j,2}$.

  Edge Server A_j:

  - Collect the responses from at least t_j clients (denote with $N_{j,3}$ this set of clients); Otherwise, aborts.
  - Verify $PoK_{i,j}$ and aborts if $PoK_{i,j}$ is failed.
  - For each $i \in N_{j,1} \setminus N_{j,2}$, calculate $\alpha_{i,j} = S.Recon(\{\alpha_{im,j}\}_{m \in N_{j,3}}, t_j)$, $r_{im,j} = KA.agree(g^{\alpha_{i,j}}, g^{\alpha_{m,j}})$ for $m \in N_{j,2}$, and $r_{i,j} = \sum_{i<m} PRG(r_{im,j}) - \sum_{m<i} PRG(r_{mi,j})$.
  - For each $i \in N_{j,2}$, calculate $\gamma_{i,j} = S.Recon(\{\gamma_{im,j}\}_{m \in N_{j,3}}, t_j)$.
  - Compute $z_j = \sum_{i \in N_{j,2}} x_{i,j} = \sum_{i \in N_{j,2}} X_{i,j} - \sum_{i \in N_{j,2}} \gamma_{i,j} + \sum_{i \in N_{j,1} \setminus N_{j,2}} r_{i,j}$.
  - Compute $\sigma_j = \prod_{i \in N_{j,2}} \hat{e}(\sigma_{i,j}, RK_{i,j})$.
  - Encrypt $D_j = Enc(KA.agree(g^b, g^{s_j}), z_j || \sigma_j || \{r_{i,j}\}_{i \in N_{j,1} \setminus N_{j,2}})$ and send D_j to the central server.

  Cloud Server A:

  - Receive the aggregated model gradients from all the edge servers.
  - For each j, decrypt D_j as $z_j || \sigma_j || \{r_{i,j}\}_{i \in N_{j,1} \setminus N_{j,2}} = Dec(KA.agree(g^b, g^{s_j}), D_j)$, and verify the validity of the aggregated gradients as $\sigma_j \hat{e}(g_1^{\sum_{i \in N_{j,1} \setminus N_{j,2}} r_{i,j}}, g^b) = \hat{e}(\prod_{i \in N_{j,2}} H(PK_{i,j} | j || g^b) g_0^{z_j}, g^b)$. Aborts if the verification is invalid.
  - Aggregate all the valid aggregated model gradients from the edge servers (denote with S this set of model gradients), as $Z = \sum_{j \in S} z_j$ and returns the global model parameters Z to the edge servers.

- **Round 3 (ModelBackward)**

  Edge Server A_j:

– Forward the global model parameters Z from the cloud server to all the clients in $N_{j,0}$.

Client $u_{i,j}$:

– Initialize the new round of local model training based on the local dataset and the received global model Z from the edge server A_j.

Finally, due to the non-guaranteed reliability of network connections in EFL environments, some clients may drop out during the execution of SVTAgg. To maintain correctness and security, SVTAgg monitors the number of active clients in each round and ensures that it remains above the secret-sharing threshold required by each edge server. Accordingly, edge servers track the number of participating clients in each training round and proceed with aggregation only when the threshold condition is satisfied.

To implement the zero-knowledge proof $PoK_{i,j}$ in a non-interactive way using Fiat-Shamir heuristic.

Prover $P(\mathsf{stmt}_{i,j}; \mathsf{wit}_{i,j})$: Given $\mathsf{wit}_{i,j} = (x_{i,j}, PRG(\gamma_{i,j}), r_{i,j}, \beta_{i,j})$ and $\mathsf{stmt}_{i,j} = (X_{i,j}, Y_{i,j}, \sigma_{i,j})$,

- Compute $h_{i,j} = H(PK_{i,j} \| j \| g^b)$ and $A_{i,j} = h_{i,j}\, g_0^{x_{i,j}} g_1^{r_{i,j}} \in G$.
- Sample fresh randomness $a_x, a_m, a_r, a_\beta \xleftarrow{\$} Z_q^*$.
- **Commitments.** Define $T_Y = g_0^{a_x + a_m + a_r}$, $T_A = g_0^{a_x} g_1^{a_r}$, $T_\sigma = A_{i,j}^{a_\beta}$.
- **Fiat–Shamir challenge.** Compute $c = H_{\mathsf{FS}}\Big(\mathsf{stmt}_{i,j} \,\|\, A_{i,j} \,\|\, T_Y \,\|\, T_A \,\|\, T_\sigma\Big) \in Z_q$.
- **Responses.** Compute

$$\begin{aligned} s_x &:= a_x + c\, x_{i,j} \pmod q, \\ s_m &:= a_m + c\, PRG(\gamma_{i,j}) \pmod q, \\ s_r &:= a_r + c\, r_{i,j} \pmod q, \\ s_\beta &:= a_\beta + c\, \beta_{i,j}^{-1} \pmod q. \end{aligned}$$

- Output the non-interactive proof $\pi_{i,j} = \big(A_{i,j},\ T_Y,\ T_A,\ T_\sigma,\ s_x,\ s_m,\ s_r,\ s_\beta\big)$.

Verifier $V(\mathsf{stmt}_{i,j}, \pi_{i,j})$: Given $\pi_{i,j} = (A_{i,j}, T_Y, T_A, T_\sigma, s_x, s_m, s_r, s_\beta)$ and $\mathsf{stmt}_{i,j} = (X_{i,j}, Y_{i,j}, \sigma_{i,j})$,

- Compute $h_{i,j} = H(PK_{i,j} \| j \| g^b)$.
- Recompute the Fiat–Shamir challenge

$$c \;:=\; H_{\mathsf{FS}}\Big(\mathsf{stmt}_{i,j} \,\|\, A_{i,j} \,\|\, T_Y \,\|\, T_A \,\|\, T_\sigma\Big) \in Z_q.$$

- Verify the following three equations:

$$\begin{aligned} g_0^{s_x+s_m+s_r} &\stackrel{?}{=} T_Y \cdot Y_{i,j}^{\,c} \\ g_0^{s_x}\, g_1^{s_r} &\stackrel{?}{=} T_A \cdot (A_{i,j}/h_{i,j})^{\,c} \\ A_{i,j}^{s_\beta} &\stackrel{?}{=} T_\sigma \cdot \sigma_{i,j}^{\,c}. \end{aligned}$$

- Accept if all checks pass.

5.4 Security Analysis

In this section, we analyze the security properties of SVTAgg, following the security goals in section 5.2.3, including security, verifiability, and traceability.

• *Security*: The security of SVTAgg is to ensure the confidentiality of each client's local model update against both semi-honest and malicious entities. To achieve this, we leverage the pairwise masking mechanism originally proposed in [26], as

$$\sum_{i=1}^{n} \tilde{\boldsymbol{w}}_i = \sum_{i=1}^{n} \boldsymbol{w}_i + \sum_{i=1}^{n} \sum_{j>i} \boldsymbol{r}_{i,j} - \sum_{i=1}^{n} \sum_{j<i} \boldsymbol{r}_{j,i}.$$

The confidentiality of client updates is based on the notion of computational indistinguishability, where an adversary cannot distinguish the observed protocol outputs from uniformly random values. Neither the edge servers nor the cloud server can infer any client's private update, even in the presence of client dropouts or partial collusion. That is, no information about the local model update $x_{i,j}$ is leaked from the masked value $X_{i,j}$, as well as the output $Y_{i,j}$.

Compared to the original secure aggregation protocol in [26], the only additional output that could potentially leak information about the client's local update is the homomorphic signature $\sigma_{i,j}$. Therefore, it is necessary to prevent any leakage of $x_{i,j}$ from $\sigma_{i,j}$. To defend against offline guessing attacks, a known vulnerability of publicly verifiable signatures,

we incorporate a random blinding factor $r_{i,j}$ into the verifiable tag generation. When $r_{i,j}$ is sampled uniformly and independently for each client, the resulting verifiable tag $\sigma_{i,j}$ is computationally indistinguishable from a random group element, thereby revealing no information about $x_{i,j}$. The uniform randomness and independence of $r_{i,j}$ have been rigorously established in Hyb5–Hyb9 of the proof of Theorem 6.3 in [26], and we omit the repetition here.

Since $g_1^{r_{i,j}}$ is never revealed, an adversary cannot remove the blinding factor from $\sigma_{i,j}$. As a result, $x_{i,j}$ cannot be inferred from $\sigma_{i,j}$, and the confidentiality of the client's local model update remains preserved.

• *Verifiability*: The verifiability is achieved based on the correctness of homomorphic signature σ_j, which is aggregated from $\sigma_{i,j}$ after proxy re-signing. If there is no client drop-out, $N_{j,1} = N_{j,2}$, so that $\sum_{i\in N_{j,2}} r_{i,j} = \sum_{i\in N_{j,2}}(\sum_{i\in N_{j,2}, m\in N_{j,2}\setminus\{i\}, i<m} PRG(r_{im,j}) - \sum_{i\in N_{j,2}, m\in N_{j,2}\setminus\{i\}, m<i} PRG(r_{mi,j})) = 0$, so that $z_j = \sum_{i\in N_{j,2}} X_{i,j} - \sum_{i\in N_{j,2}} PRG(\gamma_{i,j})$, and $\sigma_j = \prod_{i\in N_{j,2}} \hat{e}((H(PK_{i,j}||j||g^b)g_0^{x_{i,j}})^{\frac{1}{\beta_{i,j}}}, RK_{i,j})$. σ_j could be verified with the input of the aggregated local model parameters and the public key of the cloud server A. The correctness of σ_j is verified in the Equ. 5.3.

$$\begin{aligned}
\sigma_j &= \hat{e}(\prod_{i\in N_{j,2}} H(PK_{i,j}||j||g^b)g_0^{z_j}, g^b) \\
&= \hat{e}((\prod_{i\in N_{j,2}} H(PK_{i,j}||j||g^b)g_0^{\sum_{i\in N_{j,2}} x_{i,j}} g_1^{\sum_{i\in N_{j,2}} r_{i,j}})^{\frac{1}{\beta_{i,j}}}, g^{\beta_{i,j}b}) \\
&= \prod_{i\in N_{j,2}} \hat{e}((H(PK_{i,j}||j||g^b)g_0^{x_{i,j}}g_1^{r_{i,j}})^{\frac{1}{\beta_{i,j}}}, RK_{i,j}) \\
&= \prod_{i\in N_{j,2}} \hat{e}(\sigma_{i,j}, RK_{i,j}).
\end{aligned} \tag{5.3}$$

If there are some clients drop-out, $N_{j,1} \neq N_{j,2}$, the central server A needs to utilize the recovered $r_{i,j}, i \in N_{j,1}\setminus N_{j,2}$ to conduct the verification. The correctness of σ_j is verified in the Equ. 5.4.

$$
\begin{aligned}
& \sigma_j \hat{e}(g_1^{\sum_{i\in N_{j,1}\setminus N_{j,2}} r_{i,j}}, g^b) \qquad (5.4)\\
= \; & \hat{e}(\prod_{i\in N_{j,2}} H(PK_{i,j}||j||g^b) g_0^{z_j}, g^b)\\
= \; & \hat{e}(\prod_{i\in N_{j,2}} H(PK_{i,j}||j||g^b) g_0^{z_j} g_1^{\sum_{i\in N_{j,1}} r_{i,j}}, g^b)\\
= \; & \hat{e}((\prod_{i\in N_{j,2}} H(PK_{i,j}||j||g^b) g_0^{\sum_{i\in N_{j,2}} x_{i,j}} g_1^{\sum_{i\in N_{j,2}} r_{i,j}})^{\frac{1}{\beta_{i,j}}}, g^{\beta_{i,j} b}) \hat{e}(g_1^{\sum_{i\in N_{j,1}\setminus N_{j,2}} r_{i,j}}, g^b)\\
= \; & \prod_{i\in N_{j,2}} \hat{e}((H(PK_{i,j}||j||g^b) g_0^{x_{i,j}} g_1^{r_{i,j}})^{\frac{1}{\beta_{i,j}}}, RK_{i,j}) \hat{e}(g_1^{\sum_{i\in N_{j,1}\setminus N_{j,2}} r_{i,j}}, g^b)\\
= \; & \prod_{i\in N_{j,2}} \hat{e}(\sigma_{i,j}, RK_{i,j}) \hat{e}(g_1^{\sum_{i\in N_{j,1}\setminus N_{j,2}} r_{i,j}}, g^b).
\end{aligned}
$$

- *Traceability*:

To achieve traceability, SVTAgg must guarantee the integrity and authenticity of clients' local model updates. Specifically, each client generates a homomorphic signature $\sigma_{i,j}$ to authenticate its local update, while the edge server aggregates these signatures into σ_j to enable verification by the cloud server. Hence, the integrity of local updates relies on the unforgeability of both the individual homomorphic signatures and the aggregated verifiable tag. We prove the unforgeability of the individual homomorphic signatures and aggregated verifiable tag separately. We note that $r_{i,j}$ is sampled uniformly and independently for each client, which has been rigorously established in Hyb5–Hyb9 of the proof of Theorem 6.3 in [26].

a. Unforgeability of Individual Homomorphic signatures

Client $u_{i,j}$ generates a homomorphic signature as $\sigma_{i,j} = \left(H(PK_{i,j}\|j\|g^b)\, g_0^{x_{i,j}} g_1^{r_{i,j}}\right)^{\frac{1}{\beta_{i,j}}}$. The unforgeability of this signature can be reduced to the DHI assumption [22]. That is, given $h, h^s \in G$ for unknown $s \in Z_p$, no Probabilistic Polynomial-Time (PPT) adversary can compute $h^{1/s}$ with non-negligible probability.

Theorem 1: The homomorphic signature on the local model updates is existentially unforgeable against adaptive chosen message attacks under the security model [21], provided that the DHI problem in G is difficult to be addressed with a non-negligible probability in PPT.

Proof. Assume there exists a PPT adversary $\mathcal{A}$ that can break the existential unforgeability of the homomorphic signatures with non-negligible probability. We construct a simulator $\mathcal{B}$ that solves the DHI problem with non-negligible probability, thereby contradicting the hardness assumption.

Let h be a generator of the cyclic group G of prime order p. The simulator $\mathcal{B}$ is given a DHI challenge (h, h^s) for an unknown $s \in Z_p$, and its goal is to compute $h^{1/s}$. The simulator $\mathcal{B}$ plays the role of the challenger and interacts with $\mathcal{A}$ as follows.

- $\mathcal{B}$ selects random values $r, r_1, r_2 \leftarrow Z_p$ and sets the public key as $PK = h^{s/r}$, the public parameters as $g_0 = h^{sr_1}$ and $g_1 = h^{sr_2}$. These parameters are forwarded to $\mathcal{A}$.
- $\mathcal{B}$ programs the random oracle $H(\cdot)$ and maintains a list to ensure consistency of responses. Upon receiving a hash query $(PK_{i,j}, j, g^b)$ from $\mathcal{A}$, $\mathcal{B}$ flips a biased coin $\theta_{i,j} \in \{0, 1\}$ and samples $\phi_{i,j} \leftarrow Z_p$.
  - If $\theta_{i,j} = 0$, $\mathcal{B}$ sets $H(PK_{i,j} \| j \| g^b) = h^{\phi_{i,j}}$.
  - Otherwise, if $\theta_{i,j} = 1$, $\mathcal{B}$ sets $H(PK_{i,j} \| j \| g^b) = h^{s\phi_{i,j}}$.

  At last, $\mathcal{B}$ adds a tuple $(PK_{i,j}, j, g^b, \theta_{i,j}, \phi_{i,j}, h_{i,j})$ to the list, and returns $h_{i,j}$ to $\mathcal{A}$.
- When $\mathcal{A}$ queries the signing oracle on $(PK_{i,j}, j, g^b, x_{i,j}, r_{i,j})$, $\mathcal{B}$ first checks whether the corresponding hash query exists. If $(PK_{i,j}, j, g^b)$ has not been queried, $\mathcal{B}$ generates the corresponding $(\theta_{i,j}, \phi_{i,j}, h_{i,j})$ for $(PK_{i,j}, j, g^b)$.
  - If $\theta_{i,j} = 0$, $\mathcal{B}$ aborts and returns failure.
  - If $\theta_{i,j} = 1$, $\mathcal{B}$ computes $\sigma_{i,j} = h^{\phi_{i,j} + r_1 x_{i,j} + r_2 r_{i,j}}$.

  Observe that $\sigma_{i,j}$ is a valid signature on $(PK_{i,j}, j, g^b, x_{i,j}, r_{i,j})$ under the public key $h^{\frac{s}{r}}$. Finally, $\mathcal{B}$ returns $\sigma_{i,j}$ and adds $(PK_{i,j}, j, g^b, x_{i,j}, r_{i,j}, \sigma_{i,j})$ to the list.
- Eventually, $\mathcal{A}$ outputs a forgery $(PK_{i,j}, j, g^b, \hat{x}_{i,j}, \hat{r}_{i,j}, \hat{\sigma}_{i,j})$ such that no signing query has been made for $(PK_{i,j}, j, g^b, \hat{x}_{i,j}, \hat{r}_{i,j})$. If the forgery is invalid, $\mathcal{B}$ aborts and returns failure. Otherwise, $\mathcal{B}$ retrieves the corresponding tuple from the hash list.
  - If $\hat{\theta}_{i,j} = 1$, $\mathcal{B}$ aborts and returns failure.
  - If $\hat{\theta}_{i,j} = 0$, then $H(PK_{i,j} \| j \| g^b) = h^{\hat{\phi}_{i,j}}$ and $\hat{\sigma}_{i,j} = h^{\hat{\phi}_{i,j} r/s} \cdot h^{r_1 \hat{x}_{i,j} + r_2 \hat{r}_{i,j}}$. Thus, $\mathcal{B}$ computes
  $$h^{1/s} = \left(\hat{\sigma}_{i,j} \cdot h^{-(r_1 \hat{x}_{i,j} + r_2 \hat{r}_{i,j})}\right)^{1/\hat{\phi}_{i,j}},$$
  solving the DHI challenge.

Therefore, if the DHI problem is hard, no PPT adversary can forge valid homomorphic signatures on clients' local model updates with non-negligible probability, completing the proof.

b. Unforgeability of Aggregated Verifiable Tag

We show that it is computationally infeasible to forge a valid aggregated edge-level model–signature pair (z_j, σ_j) in PPT under the CONF assumption [23] in the group G_T. Specifically, this means that there exists no PPT algorithm that can solve the CONF problem, i.e., given $g, g^s, g^{sv} \in G$, where $s, v \in Z_p$, to compute $\hat{e}(g,g)^v \in G_T$.

Theorem 2. The probability that any PPT adversary can generate a valid aggregated verifiable tag σ_j that is not equal to the aggregation of the individual homomorphic signatures is negligible, provided that the CONF problem is computationally hard.

Proof. Suppose there exists a PPT adversary $\mathcal{A}$ that can break the unforgeability of the aggregated verifiable tag with non-negligible probability. Then, we construct an algorithm $\mathcal{B}$ that uses $\mathcal{A}$ as a subroutine to solve the CONF problem.

Let g be a generator of G. Algorithm $\mathcal{B}$ is given $g, D = g^s, D_1 = g^{sv}$, where $s, v \in Z_p$, and its goal is to compute $D_2 = \hat{e}(g,g)^v$. Algorithm $\mathcal{B}$ simulates the challenger and is allowed to access a signing oracle $\mathcal{SO}$ that outputs valid signatures on individual local model updates. It interacts with $\mathcal{A}$ as follows.

- $\mathcal{B}$ randomly selects $\rho_{i,j} \in Z_p$ and sets the public key $PK_{i,j} = D^{\rho_{i,j}}$ and the re-signing key $RK_{i,j} = D_1^{\rho_{i,j}}$ for all $1 \le i \le N_{j,2}$ and $1 \le j \le L$, where L denotes the number of edge servers. $\mathcal{B}$ further selects random values $\gamma_0, \gamma_1 \in Z_p$ and sets $g_0 = g^{\gamma_0}$ and $g_1 = g^{\gamma_1}$. Finally, $\mathcal{B}$ sends $(\{PK_{i,j}, RK_{i,j}\}_{1 \le i \le N_{j,2}}, g, g_0, g_1)$ to $\mathcal{A}$.
- $\mathcal{A}$ may adaptively request signatures on local model updates under any public key in $\{PK_{i,j}\}$. For each query, $\mathcal{B}$ forwards the request to the signing oracle $\mathcal{SO}$, receives the signature $\sigma_{i,j}$, and returns it to $\mathcal{A}$.
- Eventually, $\mathcal{A}$ outputs an aggregated verifiable tag $\hat{\sigma}_j$ on an aggregated edge-level model $\hat{z}_j$ computed from $(PK_{i,j}, j, g^b, \hat{x}_{i,j}, \hat{r}_{i,j})$ for all $1 \le i \le N_{j,2}$. If $\hat{\sigma}_j$ is not valid, $\mathcal{B}$ aborts and reports failure. Otherwise, $\hat{\sigma}_j$ satisfies the verification equation:

$$\hat{\sigma}_j \hat{e}(g_1^{\sum_{i \in N_{j,1} \setminus N_{j,2}} \hat{r}_{i,j}}, g^v) = \hat{e}\left(\prod_{i \in N_{j,2}} h_{i,j} g_0^{\hat{z}_j},\ g^v\right),$$

where $h_{i,j} = H(PK_{i,j} \| j \| g^b)$. Let σ_j denote the honestly generated aggregated verifiable tag corresponding to $(PK_{i,j}, j, g^b, x_{i,j}, r_{i,j}, \sigma_{i,j})$ for $1 \leq i \leq N_{j,2}$, which satisfies:

$$\sigma_j \hat{e}(g_1^{\sum_{i \in N_{j,1} \setminus N_{j,2}} r_{i,j}}, g^v) = \hat{e}\left(\prod_{i \in N_{j,2}} h_{i,j}\, g_0^{z_j},\ g^v\right).$$

If $\hat{z}_j = z_j$ and $\hat{r}_{i,j} = r_{i,j}$ for all i, then $\hat{\sigma}_j = \sigma_j$. Otherwise, define $\Delta z_j = \hat{z}_j - z_j$ and $\Delta r_{i,j} = \hat{r}_{i,j} - r_{i,j}$, where at least one is nonzero.

– If $\hat{\sigma}_j \neq \sigma_j$, dividing the two verification equations yields

$$\frac{\hat{\sigma}_j}{\sigma_j} = \hat{e}\left(\prod_{i \in N_{j,2}} g_0^{\Delta z_j}\, g_1^{-\sum_{i \in N_{j,1} \setminus N_{j,2}} \Delta r_{i,j}},\ g^v\right).$$

Substituting $g_0 = g^{\gamma_0}$ and $g_1 = g^{\gamma_1}$ gives

$$\frac{\hat{\sigma}_j}{\sigma_j} = \hat{e}\left(g^{\sum_{i \in N_{j,2}} \gamma_0 \Delta z_j - \sum_{i \in N_{j,1} \setminus N_{j,2}} \gamma_1 \Delta r_{i,j}},\ g^v\right).$$

Rearranging yields

$$\hat{e}(g,g)^v = \left(\frac{\sigma_j}{\hat{\sigma}_j}\right)^{\sum_{i \in N_{j,2}} \gamma_0 \Delta z_j - \sum_{i \in N_{j,1} \setminus N_{j,2}} \gamma_1 \Delta r_{i,j}},$$

which solves the CONF problem.

– Otherwise, the pairing term equals 1, implying

$$D_2 = \hat{e}(g,g)^v = 1^{\sum_{i \in N_{j,2}} \gamma_0 \Delta z_j - \sum_{i \in N_{j,1} \setminus N_{j,2}} \gamma_1 \Delta r_{i,j}},$$

which again yields a solution to the CONF problem.

Therefore, if the CONF problem cannot be solved with non-negligible probability in probabilistic polynomial time, then no adversary can forge aggregated verifiable tag on edge-level model updates, completing the proof.

c. Inconsistency Attack Prevention

The above results guarantee that the integrity and authenticity of each client's local model update are ensured by the unforgeability of the individual homomorphic signatures

and the aggregated verifiable tags. Consequently, any client that contributes to the aggregated global model can be reliably identified and traced by the central server through the verification of the re-signed aggregated tags. However, integrity alone is insufficient to guarantee traceability. A client could attempt to evade traceability by using inconsistent values when generating the masked update $X_{i,j}$ and the homomorphic signature $\sigma_{i,j}$, e.g., uploading a benign model update in $X_{i,j}$ while signing a different malicious update in $\sigma_{i,j}$. Such an attack would pass confidentiality protection but undermine verifiability and traceability.

To prevent this inconsistency attack, SVTAgg requires each client to generate a zero-knowledge proof of knowledge $PoK_{i,j}$ demonstrating that the same local model update $x_{i,j}$ is used consistently in both the masked model $X_{i,j}$ and the homomorphic signature $\sigma_{i,j}$, without revealing $x_{i,j}$ itself.

Formally, $PoK_{i,j}$ proves knowledge of $(x_{i,j}, PRG(\gamma_{i,j}), r_{i,j}, \beta_{i,j})$ such that $X_{i,j} = x_{i,j} + PRG(\gamma_{i,j}) + r_{i,j}$, and $\sigma_{i,j} = \left(H(PK_{i,j} \| j \| g^b)\, g_0^{x_{i,j}} g_1^{r_{i,j}}\right)^{1/\beta_{i,j}}$, thereby cryptographically binding the same $x_{i,j}$ to both the encrypted update and the signature.

- *Soundness* of the zero-knowledge proof ensures that any client that successfully passes verification must have used the same $x_{i,j}$ in both constructions; otherwise, no witness satisfying the equations exists.
- *Zero-knowledge* guarantees that the edge server and the cloud server learn nothing about $x_{i,j}$ beyond the validity of the statement.
- *Completeness* ensures that honest clients are always accepted.

Therefore, the zero-knowledge proof enforces consistency between $X_{i,j}$ and $\sigma_{i,j}$ while preserving confidentiality, closing the final loophole for traceability. As a result, any misbehaving client that injects corrupted updates can be cryptographically identified and held accountable by the central server, completing the security guarantees of SVTAgg.

5.5 Performance Evaluation

In this section, we conduct a comprehensive evaluation of SVTAgg, focusing on three critical dimensions: model accuracy, computational efficiency, and communication overhead.

5.5.1 Experimental Setup

To evaluate the effectiveness and efficiency of SVTAgg, we conduct comprehensive experiments under an EFL setting. All experiments are performed using ResNet-18 on the EMNIST dataset, which consists of handwritten digits and letters. To emulate practical EFL scenarios, the dataset is partitioned following the default EMNIST federated split, resulting in a non-IID data distribution across clients. This configuration reflects heterogeneous user data distributions commonly observed in real-world deployments.

The EFL architecture consists of multiple clients, several edge servers, and a cloud server. Clients perform local model training using SGD and upload masked local gradients along with homomorphic tags. Edge servers conduct secure edge-level aggregation and proxy re-signature operations before forwarding aggregated results to the cloud server for global aggregation and verification. The implementation is under the elliptic curve, including the operations of zero-knowledge proofs.

All experiments are conducted on a server equipped with an Intel Xeon Gold 6242R CPU (3.10 GHz) and an NVIDIA RTX A6000 GPU. The GPU is used exclusively for accelerating local model training, while all cryptographic operations, including pairwise double masking, homomorphic signature generation, zero-knowledge proof verification, and proxy re-signature, are executed on the CPU. This separation ensures that the reported cryptographic overhead reflects realistic deployment conditions in resource-constrained edge environments.

We instantiate the bilinear groups using the BLS12-381 pairing-friendly elliptic curve. Let p be the 381-bit base-field prime and the prime subgroup order is 255bits. The group $\mathbb{G}_1$ is the order-r subgroup of the elliptic curve $E_1(\mathbb{F}_p) : y^2 = x^3 + 4$, whereas $\mathbb{G}_2$ is the order-r subgroup of its twist $E_2(\mathbb{F}_{p^2}) : y^2 = x^3 + 4(u + 1)$ where $u^2 = -1$. The detailed values of p and r can be found in standard file [136]. The target group $\mathbb{G}_T$ is an order-r subgroup of $\mathbb{F}^*_{p^{12}}$. We use the optimal Ate pairing $\hat{e} : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$. BLS12-381 has embedding degree 12 and provides approximately 128-bit classical security.

5.5.2 Model Accuracy

Fig. 5.2 presents the test accuracy comparison between standard EFL and EFL integrated with SVTAgg under 500 participating clients. The number of edge servers are 5 and each edge server coordinates 100 clients. The client drop-out is not considered here. The results show that both schemes exhibit nearly identical convergence trends. The global model accuracy increases steadily during training and stabilizes at approximately 93%.

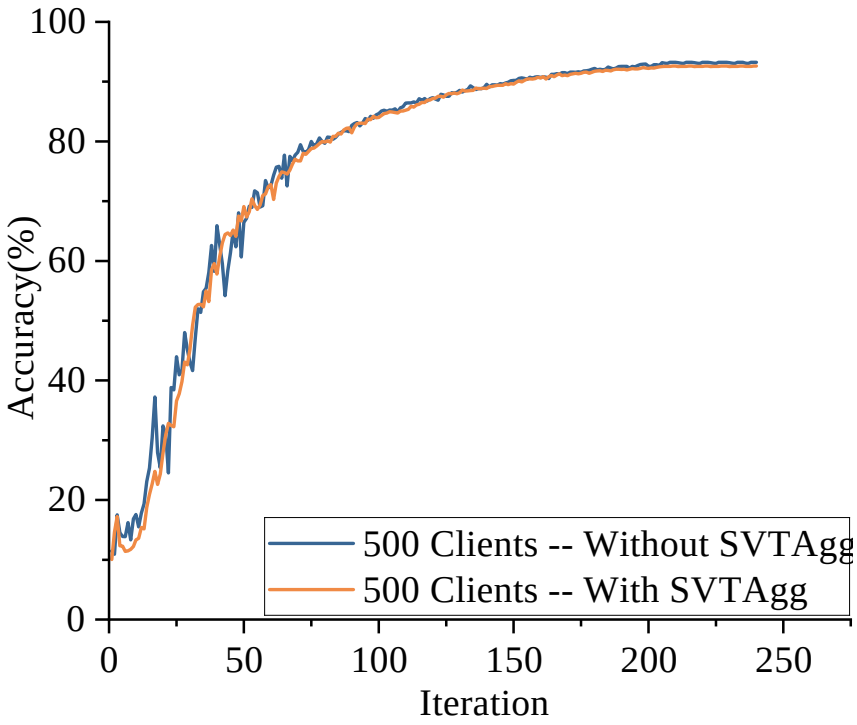


Figure 5.2: Model Accuracy with SVTAgg and without SVTAgg.

Importantly, the integration of security, verifiability, and traceability mechanisms does not introduce observable degradation in learning performance. The overlap between the "With SVTAgg" and "Without SVTAgg"" curves demonstrates that the proposed cryptographic enhancements are orthogonal to the optimization process. This confirms that SVTAgg preserves the statistical efficiency of EFL while strengthening security guarantees.

Fig. 5.3 further illustrates model accuracy alongside computational cost during training. The accuracy curve increases smoothly as training progresses, indicating stable convergence behavior. No oscillation or instability is observed due to the incorporation of secure aggregation and verification procedures. These results validate that SVTAgg maintains the effectiveness of collaborative model training in EFL systems.

Overall, the experimental results confirm that SVTAgg achieves strong security properties without sacrificing model utility, making it practical for secure EFL deployments.

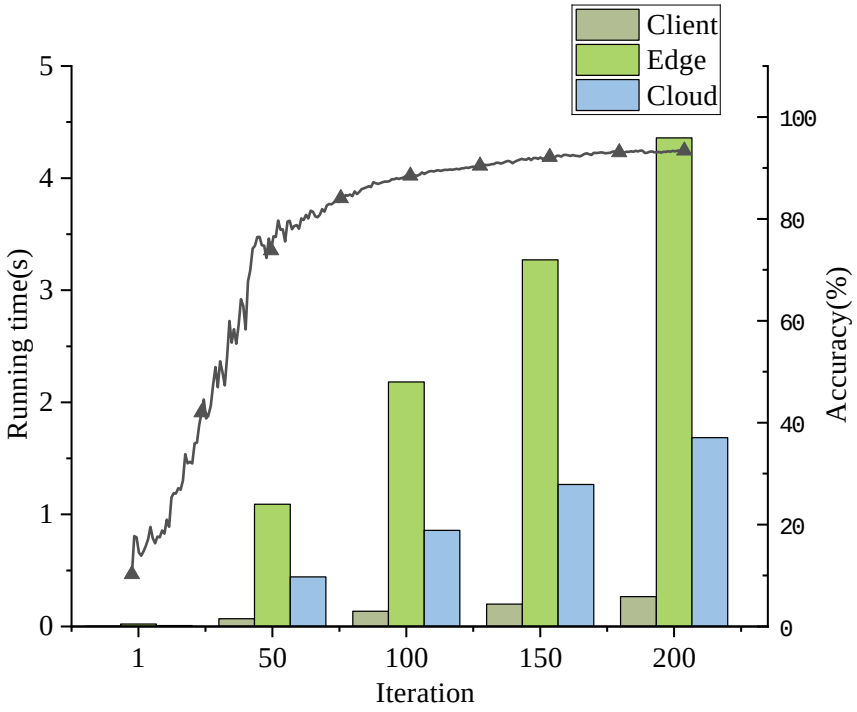


Figure 5.3: Model Accuracy and Computational Cost of Model Training of SVTAgg.

5.5.3 Computational Overhead Analysis

We evaluate the computational overhead incurred by SVTAgg at the client, edge, and cloud levels.

1) Round Computational Cost. Fig. 5.3 shows the runtime distribution across different entities during model training of SVTAgg. The number of clients is 100, and an edge server coordinates the training process of clients. The client-side cost includes local masking, homomorphic signature generation, and zero-knowledge proof construction. The edge server cost primarily stems from proof verification, aggregation, and proxy re-signature operations. The cloud server cost involves aggregated tag verification and global model aggregation. First, the running time of the client, the edge server, and the cloud server cumulatively increases as the number of training iterations grows, since each round introduces additional cryptographic and aggregation operations. Then, the results for each entity indicate that: 1) Client-side cryptographic overhead remains relatively small compared to local training time. 2) Edge servers incur higher overhead due to aggregation and verification responsibilities. 3) Cloud server overhead remains modest because it processes

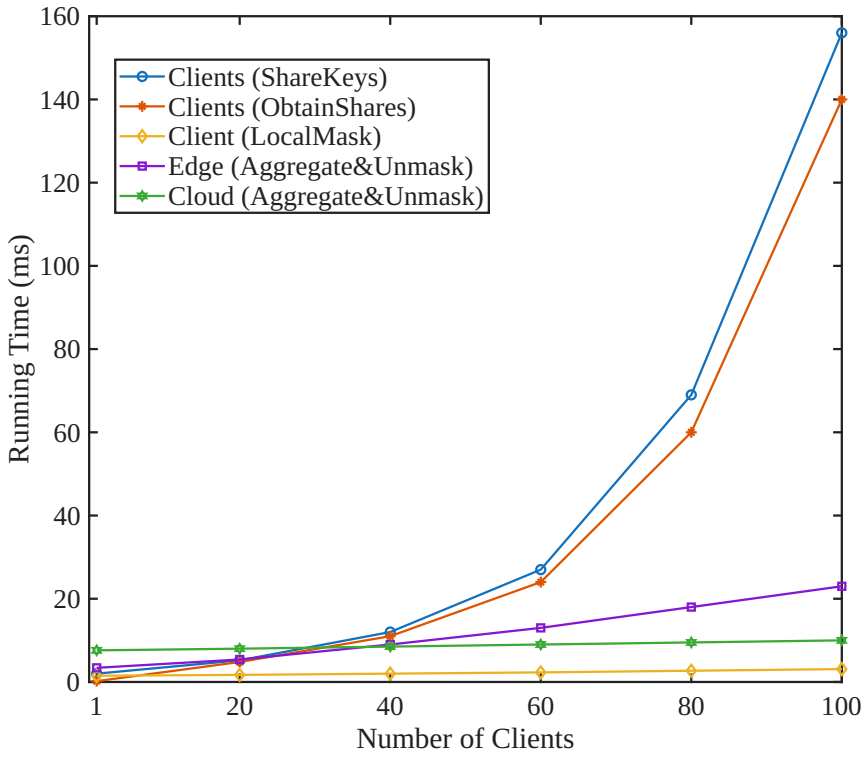


Figure 5.4: Computational Cost of Different Parties in SVTAgg.

only aggregated results rather than individual client updates. Despite the additional cryptographic operations, the overall runtime per training round remains within a few seconds, demonstrating the practical feasibility of SVTAgg.

2) Scalability with Respect to Client Population. Fig. 5.4 illustrates the computational cost as the number of clients associated with a single edge server increases. In this setting, up to 100 clients are coordinated by one edge server, which performs intermediate aggregation, while the cloud server verifies the aggregated local models and distributes the validated global model. On the client side, the costs of the ShareKeys and ObtainShares phases grow approximately quadratically with the number of participating clients. This trend is expected due to the pairwise double-masking and secret sharing mechanisms. Specifically, each client must generate secret shares for other clients and establish shared keys with all other clients within the same edge domain. As the client population increases, the number of pairwise interactions rises quadratically, leading to a noticeable increase in computational overhead. However, the operations are simple and the cost is low even there are 100 clients in each edge server. In addition, during the LocalMask (model train-

ing) round, each client generates its protected local update, homomorphic signature, and zero-knowledge proof independently. Therefore, this portion of the cost remains essentially unaffected by the number of clients. At the edge server, the Aggregate&Unmask round also scales approximately linearly with the client population. This behavior aligns with the need to verify zero-knowledge proofs, reconstruct secret shares when necessary, and aggregate masked gradients and verifiable tags. Although the edge server bears a heavier workload than individual clients, the growth rate remains moderate, demonstrating good scalability within a single edge domain. At the cloud server, the computational cost remains nearly constant as the number of clients increases. Since the cloud only processes aggregated model updates and aggregated homomorphic verifiable tags from each edge server, its workload is largely independent of the underlying client population. Only a slight increase is observed due to verification of aggregated verifiable tags. Overall, the cryptographic operations introduced by the model training phase of SVTAgg exhibit near-linear scaling rather than quadratic growth with respect to the number of clients. This confirms that the protocol design is efficient and suitable for large-scale EFL deployments.

3) Performance Impact of Client Drop-out. Client dropout affects the verification process at the cloud server, particularly in the validation of aggregated homomorphic verifiable tags. When some clients drop out, the edge server must reconstruct and remove the corresponding mask values $r_{i,j}$ for $i \in N_{j,1} \setminus N_{j,2}$. However, this additional overhead is minimal, as the removal operation only involves simple addition operations and does not require expensive cryptographic computations. Therefore, the more noticeable performance impact of client dropout appears at the cloud server during aggregated tag verification. Fig. 5.5 presents the cloud-side computational cost under varying client dropout rates (0% to 20%) and different numbers of edge servers. The results reveal three key observations:

- The computational overhead increases slightly as the dropout rate grows, primarily due to the additional mask compensation terms that must be incorporated into the aggregated tag verification process.
- Even at a 20% dropout rate, the performance degradation remains limited and well within a practical range, demonstrating the robustness of the protocol under dynamic participation.
- The cloud-side runtime scales moderately with the number of edge servers, since the cloud processes only aggregated model updates and aggregated signatures from each edge server, rather than individual client submissions.

Overall, these results confirm that SVTAgg maintains both efficiency and robustness in dynamic EFL environments where client availability may fluctuate across training rounds.

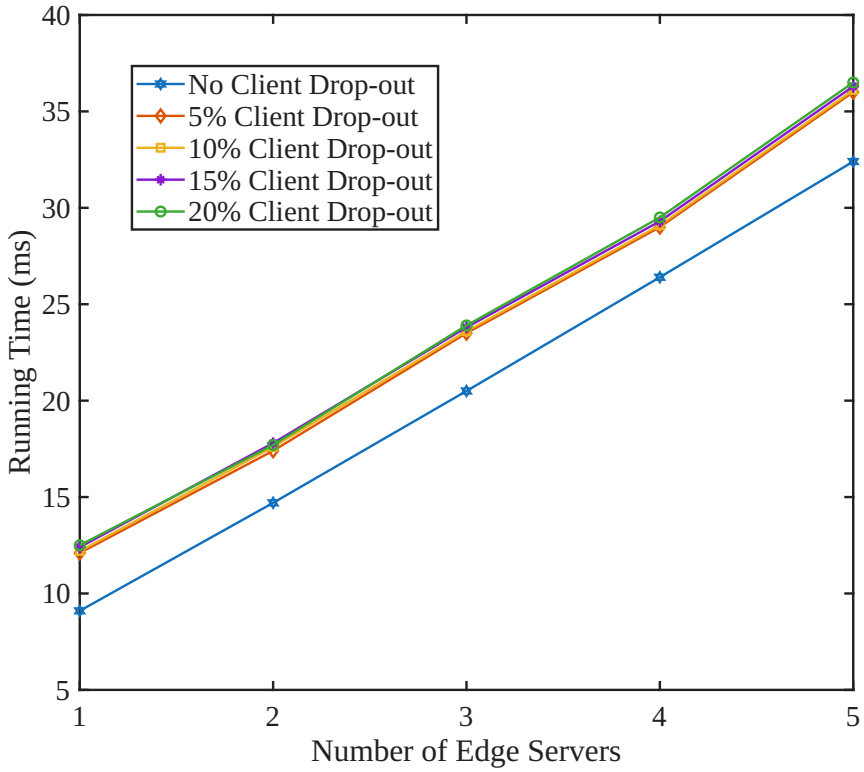


Figure 5.5: Computational Cost of Cloud Server with Client Drop-out.

4) Efficiency on Verification. Fig. 5.6 illustrates the running time of the cloud server in SVTAgg and other existing works, including VeSAFL [91], VerifyDFL [89], and FLVA [94], as the number of participating clients increases from 1 to 100. The computational cost of all schemes grows with the number of clients, since individual local updates must be processed and verified. SVTAgg aggregates client signatures and performs batch verification, whereas each signature/proof in VeSAFL [91], VerifyDFL [89], and FLVA [94] must be verified individually. SVTAgg demonstrates a nearly stable growth trend with respect to the number of clients, indicating strong scalability. Compared to VeSAFL, VerifyDFL, and FLVA, SVTAgg achieves competitive runtime performance while simultaneously providing security, verifiability, and traceability guarantees. Notably, the runtime increase remains controlled even at 100 clients, demonstrating that the integration of homomorphic signatures and verification procedures does not introduce prohibitive overhead. This confirms that SVTAgg is practical for large-scale EFL deployments.

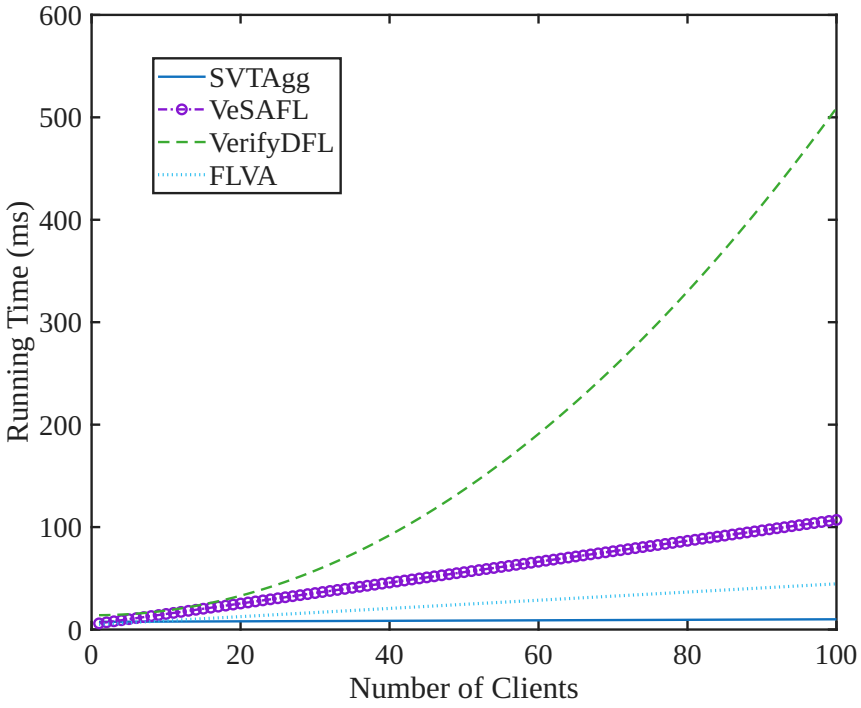


Figure 5.6: Comparison on Computational Cost of Cloud Server.

5.5.4 Communication Efficiency

In EFL, communication efficiency is a critical factor due to the hierarchical interaction among clients, edge servers, and the cloud server. In each training round, communication occurs at two primary levels: between clients and their associated edge servers, and between edge servers and the cloud server.

At the client–edge layer, each client in SVTAgg transmits its masked local model update, similar to Bonawitz et al.'s SecureAgg [26]. The dominant communication cost remains the model gradient itself, which is determined by the model architecture. For example, the parameter size of ResNet-18 is about 11.7M. In addition to the masked model gradient, the homomorphic signature $\sigma_{i,j}$ and the zero-knowledge proof $PoK_{i,j}$ are additionally transmitted from the client to the edge server. The size of $\sigma_{i,j}$ is 48 bytes for BLS12-381, and the size of $PoK_{i,j}$ is 384 bytes for one-dimensional model. At the edge–cloud layer, SVTAgg provides a more significant communication advantage compared to existing verifiable aggregation schemes. In prior works such as VeSAFL [91], VerifyDFL [89], and FLVA [94], each client's signature must be individually forwarded to the cloud

server for verification, resulting in communication overhead that scales linearly with the number of participating clients. In contrast, SVTAgg leverages the homomorphic property of signatures and integrates proxy re-signatures to support hierarchical aggregation. The edge server aggregates both the masked gradients and the corresponding verifiable tags before forwarding them to the cloud. Consequently, the cloud only receives a single aggregated model update (e.g., the size is 11.7M for ResNet-18) and a single aggregated verifiable tag per edge server (the size is 48 bytes). This reduces the communication complexity between the edge and cloud from linear in the number of clients to constant per edge server.

5.6 Summary

In this chapter, we have proposed SVTAgg, a secure, verifiable, and traceable model aggregation protocol for EFL. SVTAgg addresses key privacy and integrity challenges posed by untrusted edge servers in practical EFL deployments. Through the joint use of secret sharing, double masking, homomorphic verification, and signature-based traceability, SVTAgg simultaneously ensures gradient confidentiality, aggregation correctness, and participant accountability. Security analysis demonstrates that SVTAgg provides strong privacy, verifiability, and traceability guarantees under realistic adversarial edge settings, while experimental results show that SVTAgg achieves learning performance comparable to standard EFL with low computation and communication overhead. These results confirm that SVTAgg is both secure and practical for real-world edge learning systems. Overall, SVTAgg establishes a robust aggregation foundation for trustworthy FL at the network edge, enabling secure and privacy-preserving collaboration in emerging applications such as autonomous systems, mobile health, and large-scale IoT analytics.

However, SVTAgg primarily considers semi-honest clients and adversarial edge servers. In practical deployments, both clients and edge nodes may exhibit fully malicious or Byzantine behavior, including model poisoning, gradient manipulation, and collusion. Extending SVTAgg to detect and mitigate these attacks will be investigated in future work.

Chapter 6

Conclusion and Future Works

In this thesis, we have investigated data security and privacy for MEC. We highlight the main research contributions of this thesis and introduce several future research directions.

6.1 Main Research Contributions

In this thesis, we investigated secure, efficient, and privacy-preserving mechanisms tailored to MEC environments, with a particular focus on protecting key functions enabled by edge nodes throughout the data lifecycle at the network edge. By systematically addressing security and privacy challenges in service access, data storage, and data processing, this thesis advances the state of the art in secure edge computing. The main contributions are summarized as follows.

- To achieve efficient and fine-grained service access control in highly dynamic edge environments, we proposed E-DAC, an efficient and distributed service access control framework for MEC. In MEC, the edge nodes are delegated to provide latency-sensitive services to nearby users on behalf of the cloud service provider, which introduces new challenges in authorization, authentication, and revocation. E-DAC addresses these challenges by jointly supporting edge service authorization, service access authorization, and mutual authentication between edge servers and edge devices. Specifically, E-DAC first extends a key-aggregate cryptosystem to enable batch service authorization, allowing a service provider to aggregate authorization keys for multiple services into a constant-size aggregate key for an edge server. Second, E-DAC enables users to efficiently acquire service access authorization from the service

provider through lightweight secret sharing, avoiding excessive communication and computation overhead. Third, E-DAC supports decentralized mutual authentication between edge servers and users without relying on a centralized authority, enabling secure zero-round-trip communication and protecting service data with minimal bandwidth consumption. Moreover, E-DAC supports efficient authorization revocation for compromised or departed edge servers and users, accommodating the highly dynamic nature of MEC environments. We formally proved the security properties of E-DAC, including unforgeability of service authorization and confidentiality of service data, and conducted extensive analysis and experiments demonstrating that E-DAC achieves high computational and communication efficiency in authorization, authentication, and revocation.

- To ensure the correctness and reliability of data storage at untrusted edge servers, we proposed SecureEdge, a novel and efficient data integrity verification framework for MEC. By leveraging lightweight cryptographic primitives, SecureEdge provides an efficient platform for verifying whether edge servers correctly store and maintain end-user data. SecureEdge supports both private and public verifiability, enabling data owners and TPAs to perform integrity verification in the multiplicative group and bilinear group, respectively. It further supports multi-replica secure storage by distributing user data across multiple edge servers to improve availability and fault tolerance. To prevent malicious edge nodes from exploiting replicas stored on other nodes to falsely pass integrity verification, SecureEdge generates distinct randomized replicas for different edge servers while enabling all replicas to share the same authentication tag for proof generation. This randomization preserves data privacy and prevents TPAs from learning sensitive data during verification. In addition, SecureEdge incorporates a sequence-enforced Merkle Hash Tree to support data dynamics, allowing efficient and secure verification under data insertion, deletion, and modification. Security analysis and experimental evaluations demonstrate that SecureEdge is resilient against a wide range of data integrity attacks while maintaining low computational overhead for both authentication tag generation and integrity verification.

- To address privacy and integrity risks in collaborative edge intelligence, we proposed SVTAgg, a secure, verifiable, and traceable model aggregation protocol for EFL. Unlike conventional federated learning architectures that assume fully trusted aggregation servers, SVTAgg explicitly considers untrusted edge servers that perform intermediate aggregation. To preserve the confidentiality of local model updates, SVTAgg combines secret sharing with a double-masking mechanism, preventing information

leakage while enabling efficient aggregation. To guarantee aggregation correctness, SVTAgg introduces homomorphic verifiable tags that allow lightweight verification of intermediate aggregation results without revealing client data. Furthermore, SVTAgg supports participant traceability through homomorphic signatures, enabling authenticated provenance of submitted gradients and tracing client contributions and potential misbehaviors during local model training. Formal security analysis shows that SVTAgg achieves strong gradient privacy, aggregation correctness, and client traceability under realistic adversarial edge settings. Extensive experimental results demonstrate that SVTAgg achieves convergence performance comparable to standard federated learning, while significantly reducing computational and communication overhead compared to existing secure and verifiable aggregation schemes.

Overall, this thesis provides a comprehensive and systematic study of security and privacy mechanisms across the core stages of the edge-enabled data lifecycle. By integrating advanced cryptographic techniques with practical system design, the proposed frameworks significantly enhance the trustworthiness, efficiency, and scalability of mobile edge computing systems. These contributions lay a solid foundation for building secure and privacy-preserving edge services and edge intelligence in next-generation networks.

6.2 Future Research Directions

Although the proposed frameworks effectively address several critical security and privacy challenges in MEC, many open research issues remain. Promising future research directions include the following:

- Secure and Verifiable Edge Data Deletion: While this thesis focuses on secure service access control and data integrity verification, guaranteeing *secure and verifiable data deletion* at the network edge remains a critical and largely unexplored challenge. In practical MEC environments, user data may be replicated across multiple edge servers for availability and fault tolerance, making it difficult to ensure that all copies are irreversibly erased when deletion is requested. Future research may investigate cryptographic data deletion mechanisms, such as key-discard–based erasure, proof-of-erasure protocols, and verifiable deletion schemes that allow data owners or regulators to confirm that data has been permanently removed. Moreover, integrating secure deletion with machine unlearning and regulatory requirements, such as the GDPR's "right to be forgotten," presents additional challenges in distributed edge

environments. Secure deletion removes stored data and replicas, whereas machine unlearning must also eliminate the data's influence from trained models, intermediate updates, and final global model.

- Secure and Byzantine-robust Edge-assisted FL: The proposed SVTAgg framework primarily considers semi-honest clients and rational adversarial behaviors at edge servers. However, in realistic deployments, edge nodes and clients may exhibit fully malicious or Byzantine behavior, such as model poisoning, gradient manipulation, or collusion attacks. Future research should focus on designing *Byzantine-robust and verifiable aggregation protocols* tailored to edge-assisted and hierarchical FL architectures. Key challenges include achieving robustness against malicious participants while maintaining low communication and computation overhead, preserving gradient privacy, and supporting accountability and traceability. Combining cryptographic techniques with robust statistics, anomaly detection, and trust management mechanisms represents a promising direction for enabling resilient FL at the network edge.

- Secure and Reliable Edge Intelligence: As edge intelligence becomes increasingly pervasive, future systems must go beyond isolated security mechanisms and adopt a holistic approach to trust, reliability, and resilience. Future research may explore integrated frameworks that jointly address secure data acquisition, trustworthy model training, robust inference, and reliable system orchestration under dynamic and resource-constrained edge environments. In particular, defending against adversarial machine learning attacks, ensuring model robustness under non-IID and drifting data distributions, and providing reliability guarantees for latency-sensitive and safety-critical applications (e.g., autonomous driving, smart healthcare, and industrial IoT) remain open challenges. Leveraging secure hardware, cross-layer security designs, and adaptive edge intelligence management will be essential to enabling dependable and trustworthy edge AI systems.

- Quantum-secure Mobile Edge Computing: With the rapid advancement of quantum computing, traditional cryptographic mechanisms used in MEC may no longer provide sufficient protection against emerging quantum-enabled attacks. Future research will investigate quantum-secure MEC to protect distributed edge services. The primary objective is to integrate post-quantum cryptographic primitives into edge access control, data integrity verification, and secure learning while considering the limited computational, communication, and energy resources of mobile and edge devices.

In conclusion, security and privacy will remain foundational requirements for the continued evolution of MEC and edge intelligence. The research contributions presented in this thesis

establish essential building blocks for secure service access, trustworthy data storage, and privacy-preserving collaborative learning at the edge. Future advancements in secure data deletion, Byzantine-robust FL, reliable edge intelligence, and Quantum-secure MEC will further strengthen the trustworthiness and practicality of edge-enabled systems, enabling their deployment in increasingly complex and mission-critical real-world scenarios.

References

[1] Rodrigo Roman, Javier Lopez, and Masahiro Mambo. Mobile edge computing, fog et al.: A survey and analysis of security threats and challenges. *Future Generation Computer Systems*, 78:680–698, 2018.

[2] Milan Patel, Brian Naughton, Caroline Chan, Nurit Sprecher, Sadayuki Abeta, Adrian Neal, et al. Mobile-edge computing introductory technical white paper. *White paper, Mobile-edge Computing (MEC) Industry Initiative*, 29:854–864, 2014.

[3] Jianbing Ni, Kuan Zhang, Xiaodong Lin, and Xuemin Shen. Securing fog computing for internet of things applications: Challenges and solutions. *IEEE Communications Surveys & Tutorials*, 20(1):601–628, 2017.

[4] Vivek Kumar Sehgal, Anubhav Patrick, Ashutosh Soni, and Lucky Rajput. Smart human security framework using internet of things, cloud and fog computing. In *Intelligent Distributed Computing*, pages 251–263. Springer, 2015.

[5] Yinhao Xiao, Yizhen Jia, Chunchi Liu, Xiuzhen Cheng, Jiguo Yu, and Weifeng Lv. Edge computing security: State of the art and challenges. *Proceedings of the IEEE*, 107(8):1608–1631, 2019.

[6] Jianbing Ni, Xiaodong Lin, Kuan Zhang, and Xuemin Shen. Privacy-preserving real-time navigation system using vehicular crowdsourcing. In *2016 IEEE 84th vehicular technology Conference (VTC-Fall)*, pages 1–5. IEEE, 2016.

[7] Hao Kong, Cheng Huang, Jiadi Yu, and Xuemin Shen. A survey of mmwave radar-based sensing in autonomous vehicles, smart homes and industry. *IEEE Communications Surveys & Tutorials*, 27(1):463–508, 2024.

[8] Xuehan Ye, Kaige Qu, Weihua Zhuang, and Xuemin Shen. Accuracy-aware cooperative sensing and computing for connected autonomous vehicles. *IEEE Transactions on Mobile Computing*, 23(8):8193–8207, 2023.

[9] Yingying Pei, Mushu Li, Xinyu Huang, and Xuemin Shen. Qoe-aware volumetric video caching and rendering for mobile extended reality services. *IEEE Internet of Things Journal*, 2025.

[10] Xu Zhang, Hao Chen, Yangchao Zhao, Zhan Ma, Yiling Xu, Haojun Huang, Hao Yin, and Dapeng Oliver Wu. Improving cloud gaming experience through mobile edge computing. *IEEE Wireless Communications*, 26(4):178–183, 2019.

[11] Sarah Choksi, Skyler Szot, Chengbo Zang, Kaan Yarali, Yuqing Cao, Feroz Ahmad, Zixuan Xiang, Daniel P Bitner, Zoran Kostic, and Filippo Filicori. Bringing artificial intelligence to the operating room: edge computing for real-time surgical phase recognition. *Surgical Endoscopy*, 37(11):8778–8784, 2023.

[12] Peng Huang, Liekang Zeng, Xu Chen, Ke Luo, Zhi Zhou, and Shuai Yu. Edge robotics: Edge-computing-accelerated multirobot simultaneous localization and mapping. *IEEE Internet of Things Journal*, 9(15):14087–14102, 2022.

[13] Tie Qiu, Jiancheng Chi, Xiaobo Zhou, Zhaolong Ning, Mohammed Atiquzzaman, and Dapeng Oliver Wu. Edge computing in industrial internet of things: Architecture, advances and challenges. *IEEE Communications Surveys & Tutorials*, 22(4):2462–2488, 2020.

[14] Qiang He, Zhaolin Xi, Zheng Feng, Yueyang Teng, Lianbo Ma, Yuliang Cai, and Keping Yu. Telemedicine monitoring system based on fog/edge computing: A survey. *IEEE Transactions on Services Computing*, 18(1):479–498, 2024.

[15] Latif U Khan, Ibrar Yaqoob, Nguyen H Tran, SM Ahsan Kazmi, Tri Nguyen Dang, and Choong Seon Hong. Edge-computing-enabled smart cities: A comprehensive survey. *IEEE Internet of Things Journal*, 7(10):10200–10232, 2020.

[16] Javad Roostaei, Yongli Z Wager, Weisong Shi, Timothy Dittrich, Carol Miller, and Kishore Gopalakrishnan. Iot-based edge computing (iotec) for improved environmental monitoring. *Sustainable Computing: Informatics and Systems*, 38:1–7, 2023.

[17] Yutao Huang, Yuhe Lu, Feng Wang, Xiaoyi Fan, Jiangchuan Liu, and Victor CM Leung. An edge computing framework for real-time monitoring in smart grid. In *2018 IEEE International conference on industrial internet (ICII)*, pages 99–108. IEEE, 2018.

[18] Lina A Haibeh, Mustapha CE Yagoub, and Abdallah Jarray. A survey on mobile edge computing infrastructure: Design, resource management, and optimization approaches. *IEEE Access*, 10:27591–27610, 2022.

[19] Yoshinori Aono, Takuya Hayashi, Lihua Wang, Shiho Moriai, et al. Privacy-preserving deep learning via additively homomorphic encryption. *IEEE Transactions on Information Forensics and Security*, 13(5):1333–1345, 2017.

[20] Jiaqi Zhao, Hui Zhu, Fengwei Wang, Rongxing Lu, Zhe Liu, and Hui Li. Pvd-fl: A privacy-preserving and verifiable decentralized federated learning framework. *IEEE Transactions on Information Forensics and Security*, 17:2059–2073, 2022.

[21] Dan Boneh, Ben Lynn, and Hovav Shacham. Short signatures from the weil pairing. *Journal of Cryptology*, 17(4):297–319, 2004.

[22] Benoît Libert and Jean-Jacques Quisquater. Improved signcryption from q-diffie-hellman problems. In *Proc. International Conference on Security in Communication Networks*, pages 220–234. Springer, 2004.

[23] Chih-Hung Li and Josef Pieprzyk. Conference key agreement from secret sharing. In *Proc. Australasian Conference on Information Security and Privacy*, pages 64–76. Springer, 1999.

[24] Xu Han, Dawei Zhang, Zongmin Huang, Shuang Yao, and Zuodong Wu. Revocable one-time ring signature from pairings. *Wireless Communications and Mobile Computing*, 2022(1):8021267, 2022.

[25] Brent Waters. Functional encryption for regular languages. In *Annual Cryptology Conference*, pages 218–235. Springer, 2012.

[26] Keith Bonawitz, Vladimir Ivanov, Ben Kreuter, Antonio Marcedone, H Brendan McMahan, Sarvar Patel, Daniel Ramage, Aaron Segal, and Karn Seth. Practical secure aggregation for privacy-preserving machine learning. In *Proc. 2017 ACM SIGSAC Conference on Computer and Communications Security (ACM CCS)*, pages 1175–1191, 2017.

[27] Yichen Hou, Sahil Garg, Lin Hui, Dushantha Nalin K Jayakody, Rui Jin, and M Shamim Hossain. A data security enhanced access control mechanism in mobile edge computing. *IEEE Access*, 8:136119–136130, 2020.

[28] Ruei-Hau Hsu, Jemin Lee, Tony QS Quek, and Jyh-Cheng Chen. Reconfigurable security: Edge-computing-based framework for iot. *IEEE Network*, 32(5):92–99, 2018.

[29] Xiong Li, Tian Liu, Chaoyang Chen, Qingfeng Cheng, Xiaosong Zhang, and Neeraj Kumar. A lightweight and verifiable access control scheme with constant size ciphertext in edge-computing-assisted iot. *IEEE Internet of Things Journal*, 9(19):19227–19237, 2022.

[30] Chaosheng Feng, Keping Yu, Moayad Aloqaily, Mamoun Alazab, Zhihan Lv, and Shahid Mumtaz. Attribute-based encryption with parallel outsourced decryption for edge intelligent iov. *IEEE Transactions on Vehicular Technology*, 69(11):13784–13795, 2020.

[31] Xiang Li, Hui Tian, and Jianting Ning. Secure online/offline attribute-based encryption for iot users in cloud computing. In *2019 Provable Security: 13th International Conference (ProvSec)*, pages 347–354. Springer, 2019.

[32] Zhishuo Zhang, Guanjie Huang, Sunqiang Hu, Wei Zhang, Yi Wu, and Zhiguang Qin. Fdo-abe: A fully decentralized lightweight access control architecture for mobile edge computing. In *2021 IEEE 6th International Conference on Computer and Communication Systems (ICCCS)*, pages 193–198, 2021.

[33] Yaodong Huang, Jiarui Zhang, Jun Duan, Bin Xiao, Fan Ye, and Yuanyuan Yang. Resource allocation and consensus of blockchains in pervasive edge computing environments. *IEEE Transactions on Mobile Computing*, 21(9):3298–3311, 2021.

[34] Ruizhe Yang, F Richard Yu, Pengbo Si, Zhaoxin Yang, and Yanhua Zhang. Integrated blockchain and edge computing systems: A survey, some research issues and challenges. *IEEE Communications Surveys & Tutorials*, 21(2):1508–1532, 2019.

[35] Linsheng Yu, Mingxing He, Hongbin Liang, Ling Xiong, and Yang Liu. A blockchain-based authentication and authorization scheme for distributed mobile cloud computing services. *Sensors*, 23(3):1–34, 2023.

[36] Sourav Saha, Durbadal Chattaraj, Basudeb Bera, and Ashok Kumar Das. Consortium blockchain-enabled access control mechanism in edge computing based generic internet of things environment. *Transactions on Emerging Telecommunications Technologies*, 32(6):1–34, 2021.

[37] Bingcheng Jiang, Qian He, Peng Liu, Sabita Maharjan, and Yan Zhang. Blockchain empowered secure video sharing with access control for vehicular edge computing. *IEEE Transactions on Intelligent Transportation Systems*, 24(9):9041–9054, 2023.

[38] Debnath Bhattacharyya, Rahul Ranjan, Farkhod Alisherov, Minkyu Choi, et al. Biometric authentication: A review. *International Journal of u-and e-Service, Science and Technology*, 2(3):13–28, 2009.

[39] Qi Jiang, Ning Zhang, Jianbing Ni, Jianfeng Ma, Xindi Ma, and Kim-Kwang Raymond Choo. Unified biometric privacy preserving three-factor authentication and key agreement for cloud-assisted autonomous vehicles. *IEEE Transactions on Vehicular Technology*, 69(9):9390–9401, 2020.

[40] Sean Dougherty, Reza Tourani, Gaurav Panwar, Roopa Vishwanathan, Satyajayant Misra, and Srikathyayani Srikanteswara. Apecs: A distributed access control framework for pervasive edge computing services. In *Proc. of ACM CCS*, pages 1405–1420, 2021.

[41] Xiaotong Zhou, Debiao He, Jianting Ning, Min Luo, and Xinyi Huang. Aadec: Anonymous and auditable distributed access control for edge computing services. *IEEE Transactions on Information Forensics and Security*, 18:290–303, 2022.

[42] Jingwei Li, Jin Li, Dongqing Xie, and Zhang Cai. Secure auditing and deduplicating data in cloud. *IEEE Transactions on Computers*, 65(8):2386–2396, 2015.

[43] Mehdi Sookhak, Abdullah Gani, Hamid Talebian, and Adnan Akhunzada. Remote data auditing in cloud computing environments: a survey, taxonomy, and open issues. *ACM Computing Surveys*, 47(4):1–34, 2015.

[44] Yuan Zhang, Chunxiang Xu, Xiaohui Liang, Hongwei Li, Yi Mu, and Xiaojun Zhang. Efficient public verification of data integrity for cloud storage systems from indistinguishability obfuscation. *IEEE Transactions on Information Forensics and Security*, 12(3):676–688, 2016.

[45] Wenting Shen, Jing Qin, Jia Yu, Rong Hao, Jiankun Hu, and Jixin Ma. Data integrity auditing without private key storage for secure cloud storage. *IEEE Transactions on Cloud Computing*, 9(4):1408–1421, 2019.

[46] Jianbing Ni, Kuan Zhang, Yong Yu, and Tingting Yang. Identity-based provable data possession from rsa assumption for secure cloud storage. *IEEE Transactions on Dependable and Secure Computing*, 19(3):1753–1769, 2020.

[47] Tao Jiang, Xiaofeng Chen, and Jianfeng Ma. Public integrity auditing for shared dynamic cloud data with group user revocation. *IEEE Transactions on Computers*, 65(8):2363–2373, 2015.

[48] Yao Zhao, Youyang Qu, Yong Xiang, Md Palash Uddin, Dezhong Peng, and Longxiang Gao. A comprehensive survey on edge data integrity verification: Fundamentals and future trends. *ACM Computing Surveys*, 57(1):1–34, 2024.

[49] Cheng Wang, Zenghui Yuan, Pan Zhou, Zichuan Xu, Ruixuan Li, and Dapeng Oliver Wu. The security and privacy of mobile-edge computing: An artificial intelligence perspective. *IEEE Internet of Things Journal*, 10(24):22008–22032, 2023.

[50] Mingxi Liu, Ning Lu, Yihong Wen, Qingfeng Cheng, and Wenbo Shi. Sea: Secure and efficient public auditing for edge-assisted iot aggregated data sharing. *Mobile Networks and Applications*, 29(5):1477–1488, 2024.

[51] Bo Li, Qiang He, Feifei Chen, Hai Jin, Yang Xiang, and Yun Yang. Auditing cache data integrity in the edge computing environment. *IEEE Transactions on Parallel and Distributed Systems*, 32(5):1210–1223, 2020.

[52] Bo Li, Qiang He, Feifei Chen, Hai Jin, Yang Xiang, and Yun Yang. Inspecting edge data integrity with aggregate signature in distributed edge computing environment. *IEEE Transactions on Cloud Computing*, 10(4):2691–2703, 2021.

[53] Guangming Cui, Qiang He, Bo Li, Xiaoyu Xia, Feifei Chen, Hai Jin, Yang Xiang, and Yun Yang. Efficient verification of edge data integrity in edge computing environment. *IEEE Transactions on Services Computing*, 15(6):3233–3244, 2021.

[54] Md Rashedul Islam, Yong Xiang, Md Palash Uddin, Yao Zhao, Jonathan Kua, and Longxiang Gao. Multiple edge data integrity verification with multi-vendors and multi-servers in mobile edge computing. *IEEE Transactions on Mobile Computing*, 24(6):4668–4683, 2025.

[55] Wei Tong, Wenjie Chen, Bingbing Jiang, Fengyuan Xu, Qun Li, and Sheng Zhong. Privacy-preserving data integrity verification for secure mobile edge storage. *IEEE Transactions on Mobile Computing*, 22(9):5463–5478, 2022.

[56] Zhuoqun Yan, Wenfang Zhang, Xiaomin Wang, and Muhammad Khurram Khan. Multidimensional data integrity checking scheme for iot-edge computing-assisted intelligent railway systems. *IEEE Transactions on Vehicular Technology*, 74(5):7263–7277, 2025.

[57] Zahra Seyedi, Farhad Rahmati, Mohammad Ali, and Ximeng Liu. Decentralized data integrity inspection offloading in edge computing systems using potential games. *IEEE Transactions on Mobile Computing*, 24(5):3950–3961, 2024.

[58] Bo Li, Qiang He, Feifei Chen, Haipeng Dai, Hai Jin, Yang Xiang, and Yun Yang. Cooperative assurance of cache data integrity for mobile edge computing. *IEEE Transactions on Information Forensics and Security*, 16:4648–4662, 2021.

[59] Bo Li, Qiang He, Liang Yuan, Feifei Chen, Lingjuan Lyu, and Yun Yang. Edgewatch: Collaborative investigation of data integrity at the edge based on blockchain. In *Proc. of ACM KDD*, pages 3208–3218, Washington DC, August 14-18, 2022.

[60] Yao Zhao, Youyang Qu, Feifei Chen, Yong Xiang, and Longxiang Gao. Data integrity verification in mobile edge computing with multi-vendor and multi-server. *IEEE Transactions on Mobile Computing*, 23(5):5418–5432, 2023.

[61] James Henry Bell, Kallista A Bonawitz, Adrià Gascón, Tancrède Lepoint, and Mariana Raykova. Secure single-server aggregation with (poly) logarithmic overhead. In *Proc. 2020 ACM SIGSAC Conference on Computer and Communications Security (ACM CCS)*, pages 1253–1269, 2020.

[62] Jinhyun So, Başak Güler, and A Salman Avestimehr. Turbo-aggregate: Breaking the quadratic aggregation barrier in secure federated learning. *IEEE Journal on Selected Areas in Information Theory*, 2(1):479–489, 2021.

[63] Swanand Kadhe, Nived Rajaraman, O Ozan Koyluoglu, and Kannan Ramchandran. Fastsecagg: Scalable secure aggregation for privacy-preserving federated learning. *arXiv preprint arXiv:2009.11248*, 2020.

[64] Jinhyun So, Chaoyang He, Chien-Sheng Yang, Songze Li, Qian Yu, Ramy E Ali, Basak Guler, and Salman Avestimehr. Lightsecagg: a lightweight and versatile design for secure aggregation in federated learning. *Proceedings of Machine Learning and Systems*, 4:694–720, 2022.

[65] Fucai Luo, Saif Al-Kuwari, Haiyan Wang, and Xingfu Yan. Fssa: Efficient 3-round secure aggregation for privacy-preserving federated learning. *arXiv preprint arXiv:2305.12950*, 2023.

[66] Hossein Fereidooni, Samuel Marchal, Markus Miettinen, Azalia Mirhoseini, Helen Möllering, Thien Duc Nguyen, Phillip Rieger, Ahmad-Reza Sadeghi, Thomas Schneider, Hossein Yalame, et al. Safelearn: Secure aggregation for private federated learn-

ing. In *Proc. 2021 IEEE Security and Privacy Workshops (SPW)*, pages 56–62. IEEE, 2021.

[67] Jinling Tang, Haixia Xu, Mingsheng Wang, Tao Tang, Chunying Peng, and Huimei Liao. A flexible and scalable malicious secure aggregation protocol for federated learning. *IEEE Transactions on Information Forensics and Security*, 19:4174–4187, 2024.

[68] Runhua Xu, Bo Li, Chao Li, James BD Joshi, Shuai Ma, and Jianxin Li. Tapfed: Threshold secure aggregation for privacy-preserving federated learning. *IEEE Transactions on Dependable and Secure Computing*, 21(5):4309–4323, 2024.

[69] Pau-Chen Cheng, Kevin Eykholt, Zhongshu Gu, Hani Jamjoom, KR Jayaram, Enriquillo Valdez, and Ashish Verma. Deta: minimizing data leaks in federated learning via decentralized and trustworthy aggregation. In *Proc. 19th European Conference on Computer Systems*, pages 219–235, 2024.

[70] Aditya Pribadi Kalapaaking, Ibrahim Khalil, Mohammad Saidur Rahman, Mohammed Atiquzzaman, Xun Yi, and Mahathir Almashor. Blockchain-based federated learning with secure aggregation in trusted execution environment for internet-of-things. *IEEE Transactions on Industrial Informatics*, 19(2):1703–1714, 2022.

[71] Xue Yang, Zifeng Liu, Xiaohu Tang, Rongxing Lu, and Bo Liu. An efficient and multi-private key secure aggregation scheme for federated learning. *IEEE Transactions on Services Computing*, 17(5):1998–2011, 2024.

[72] Jingcheng Song, Weizheng Wang, Thippa Reddy Gadekallu, Jianyu Cao, and Yining Liu. Eppda: An efficient privacy-preserving data aggregation federated learning scheme. *IEEE Transactions on Network Science and Engineering*, 10(5):3047–3057, 2022.

[73] Zisang Xu, Ruirui Zhang, Wei Liang, Kuan-Ching Li, Ke Gu, Xiong Li, and Jialun Huang. A privacy-preserving data aggregation protocol for internet of vehicles with federated learning. *IEEE Transactions on Intelligent Vehicles*, 10(1):217–227, 2024.

[74] Sizai Hou, Songze Li, Tayyebeh Jahani-Nezhad, and Giuseppe Caire. Priroagg: Achieving robust model aggregation with minimum privacy leakage for federated learning. *IEEE Transactions on Information Forensics and Security*, 20:5690–5704, 2025.

[75] Renwen Ma, Kai Hwang, Mo Li, and Yiming Miao. Trusted model aggregation with zero-knowledge proofs in federated learning. *IEEE Transactions on Parallel and Distributed Systems*, 35(11):2284–2296, 2024.

[76] Zhuosheng Zhang, Jiarui Li, Shucheng Yu, and Christian Makaya. Safelearning: Secure aggregation in federated learning with backdoor detectability. *IEEE Transactions on Information Forensics and Security*, 18:3289–3304, 2023.

[77] Meng Shen, Jing Wang, Jie Zhang, Qinglin Zhao, Bohan Peng, Tong Wu, Liehuang Zhu, and Ke Xu. Secure decentralized aggregation to prevent membership privacy leakage in edge-based federated learning. *IEEE Transactions on Network Science and Engineering*, 11(3):3105–3119, 2024.

[78] Tantan Zhao, Fan Li, and Lijun He. Drl-based secure aggregation and resource orchestration in mec-enabled hierarchical federated learning. *IEEE Internet of Things Journal*, 10(20):17865–17880, 2023.

[79] Yuanyuan He, Jie Zhang, Peng Yang, Zhe Sun, and Xuemin Shen. Ppbr: Privacy-preserving and byzantine-robust edge-assisted hierarchical federated learning in mobile networks. *IEEE Transactions on Mobile Computing*, 25(2):1595–1612, 2026.

[80] Shuyang Wang, Hongcheng Xie, Yu Guo, Fangda Guo, Fangming Jing, and Rongfang Bie. Privacy-preserving and efficient model aggregation in edge-assisted federated learning. In *Proc. International Conference on Database Systems for Advanced Applications*, pages 511–521. Springer, 2024.

[81] Lingling Wang, Mei Huang, Zhengyin Zhang, Meng Li, Jingjing Wang, and Keke Gai. Rasa: Robust and adaptive secure aggregation for edge-assisted hierarchical federated learning. *IEEE Transactions on Information Forensics and Security*, 20:4280–4295, 2025.

[82] Guowen Xu, Hongwei Li, Sen Liu, Kan Yang, and Xiaodong Lin. Verifynet: Secure and verifiable federated learning. *IEEE Transactions on Information Forensics and Security*, 15:911–926, 2019.

[83] Xiaojie Guo, Zheli Liu, Jin Li, Jiqiang Gao, Boyu Hou, Changyu Dong, and Thar Baker. Verifl: Communication-efficient and fast verifiable aggregation for federated learning. *IEEE Transactions on Information Forensics and Security*, 16:1736–1751, 2020.

[84] Baturalp Buyukates, Jinhyun So, Hessam Mahdavifar, and Salman Avestimehr. Lightverifl: A lightweight and verifiable secure aggregation for federated learning. *IEEE Journal on Selected Areas in Information Theory*, 5:285–301, 2024.

[85] Zhangshuang Guan, Yulin Zhao, Zhiguo Wan, and Jinsong Han. Opsa: Efficient and verifiable one-pass secure aggregation with tee for federated learning. *IEEE Transactions on Dependable and Secure Computing*, 22(5):5321–5334, 2025.

[86] Bingxue Zhang, Guangguang Lu, Pengpeng Qiu, Xumin Gui, and Yang Shi. Advancing federated learning through verifiable computations and homomorphic encryption. *Entropy*, 25(11):1550, 2023.

[87] Ganesh Kumar Mahato, Aiswaryya Banerjee, Swarnendu Kumar Chakraborty, and Xiao-Zhi Gao. Privacy preserving verifiable federated learning scheme using blockchain and homomorphic encryption. *Applied Soft Computing*, 167:112405, 2024.

[88] Ahmed Ayoub Bellachia, Mouhamed Amine Bouchiha, Yacine Ghamri-Doudane, and Mourad Rabah. Verifbfl: Leveraging zk-snarks for a verifiable blockchained federated learning. *arXiv preprint arXiv:2501.04319*, 2025.

[89] Shuai Wang, Youliang Tian, Jinbo Xiong, Jianfeng Ma, and Yan Zhang. Verifydfl: Secure aggregation for decentralized federated learning with input validation in mobile edge intelligence. *IEEE Transactions on Cognitive Communications and Networking*, 12:2526–2541, 2025.

[90] Yijian Zhong, Wuzheng Tan, Zhifeng Xu, Shixin Chen, Jiasi Weng, and Jian Weng. Wvfl: Weighted verifiable secure aggregation in federated learning. *IEEE Internet of Things Journal*, 11(11):19926–19936, 2024.

[91] Jaouhara Bouamama, Yahya Benkaouz, and Mohammed Ouzzif. Vesafl: Verifiable secure aggregation for privacy-preserving federated learning. *IEEE Transactions On Dependable And Secure Computing*, 22(6):6592–6609, 2025.

[92] Yushan Siriwardhana, Pawani Porambage, Madhusanka Liyanage, Samuel Marchal, and Mika Ylianttila. Shield-secure aggregation against poisoning in hierarchical federated learning. *IEEE Transactions on Dependable and Secure Computing*, 22(2):1845–1863, 2025.

[93] Lulu Wang, Mirko Polato, Alessandro Brighente, Mauro Conti, Lei Zhang, and Lin Xu. Priverifl: Privacy-preserving and aggregation-verifiable federated learning. *IEEE Transactions on Services Computing*, 18(2):998–1011, 2025.

[94] Haoran Xie, Yujue Wang, Yong Ding, Changsong Yang, Haibin Zheng, and Bo Qin. Verifiable federated learning with privacy-preserving data aggregation for consumer electronics. *IEEE Transactions on Consumer Electronics*, 70(1):2696–2707, 2023.

[95] Lingshuang Liu, Cheng Huang, Dan Zhu, Dongxiao Liu, Jianbing Ni, and Xuemin Shen. Enabling efficient and distributed access control for pervasive edge computing services. *IEEE Transactions on Mobile Computing*, 23(12):11342–11356, 2024.

[96] Shengmin Xu, Jianting Ning, Xinyi Huang, Jianying Zhou, and Robert H Deng. Server-aided bilateral access control for secure data sharing with dynamic user groups. *IEEE Transactions on Information Forensics and Security*, 16:4746–4761, 2021.

[97] Guowen Xu, Hongwei Li, Hao Ren, Xiaodong Lin, and Xuemin Shen. Dna similarity search with access control over encrypted cloud data. *IEEE Transactions on Cloud Computing*, 10(2):1233–1252, 2020.

[98] Yuanni Liu, Man Xiao, Shanzhi Chen, Fan Bai, Jianli Pan, and Di Zhang. An intelligent edge-chain-enabled access control mechanism for iov. *IEEE Internet of Things Journal*, 8(15):12231–12241, 2021.

[99] Qi Li, Bin Xia, Haiping Huang, Yinghui Zhang, and Tao Zhang. Trac: traceable and revocable access control scheme for mhealth in 5g-enabled iiot. *IEEE Transactions on Industrial Informatics*, 18(5):3437–3448, 2021.

[100] Donghyun Yu, Ruei-Hau Hsu, Jemin Lee, and Sungjin Lee. Ec-svc: Secure can bus in-vehicle communications with fine-grained access control based on edge computing. *IEEE Transactions on Information Forensics and Security*, 17:1388–1403, 2021.

[101] Hao Guo, Wanxin Li, Mark Nejad, and Chien-Chung Shen. Access control for electronic health records with hybrid blockchain-edge architecture. In *Proc. of Blockchain*, pages 44–51, 2019.

[102] Damiano Di Francesco Maesa, Paolo Mori, and Laura Ricci. Blockchain based access control. In *Proc. of DAIS*, pages 206–220, 2017.

[103] Cheng Huang, Liang Xue, Dongxiao Liu, Xuemin Shen, Weihua Zhuang, Rob Sun, and Bidi Ying. Blockchain-assisted transparent cross-domain authorization and authentication for smart city. *IEEE Internet of Things Journal*, 9(18):17194–17209, 2022.

[104] Hung Dang, Tien Tuan Anh Dinh, Dumitrel Loghin, Ee-Chien Chang, Qian Lin, and Beng Chin Ooi. Towards scaling blockchain systems via sharding. In *Proc. of ACM SIGMOD*, pages 123–140, 2019.

[105] Satyajayant Misra, Reza Tourani, Frank Natividad, Travis Mick, Nahid Ebrahimi Majd, and Hong Huang. Accconf: An access control framework for leveraging in-network cached data in the icn-enabled wireless edge. *IEEE Transactions on Dependable and Secure Computing*, 16(1):5–17, 2017.

[106] Kaiping Xue, Peixuan He, Xiang Zhang, Qiudong Xia, David SL Wei, Hao Yue, and Feng Wu. A secure, efficient, and accountable edge-based access control framework for information centric networks. *IEEE/ACM Transactions on Networking*, 27(3):1220–1233, 2019.

[107] Cheng-Kang Chu, Sherman SM Chow, Wen-Guey Tzeng, Jianying Zhou, and Robert H Deng. Key-aggregate cryptosystem for scalable data sharing in cloud storage. *IEEE Transactions on Parallel and Distributed Systems*, 25(2):468–477, 2013.

[108] Wen-Guey Tzeng and Zhi-Jia Tzeng. A public-key traitor tracing scheme with revocation using dynamic shares. In *Proc. of PKC*, pages 207–224, 2001.

[109] Sikhar Patranabis, Yash Shrivastava, and Debdeep Mukhopadhyay. Provably secure key-aggregate cryptosystems with broadcast aggregate keys for online data sharing on the cloud. *IEEE Transactions on Computers*, 66(5):891–904, 2016.

[110] Jan Camenisch and Markus Stadler. Efficient group signature schemes for large groups. In *Advances in Cryptology—CRYPTO*, pages 410–424. Springer, 1997.

[111] Yao Zhao, Youyang Qu, Yong Xiang, Feifei Chen, and Longxiang Gao. Data re-outsourcing detection with latency-constraint for edge storage. *IEEE Transactions on Services Computing*, 18(4):1996–2009, 2025.

[112] Dan Liu, Zheng Yan, Wenxiu Ding, and Mohammed Atiquzzaman. A survey on secure data analytics in edge computing. *IEEE Internet of Things Journal*, 6(3):4946–4967, 2019.

[113] Syed Noorulhassan Shirazi, Antonios Gouglidis, Arsham Farshad, and David Hutchison. The extended cloud: Review and analysis of mobile edge computing and fog from a security and resilience perspective. *IEEE Journal on Selected Areas in Communications*, 35(11):2586–2595, 2017.

[114] Meng Li, F Richard Yu, Pengbo Si, and Yanhua Zhang. Energy-efficient machine-to-machine (m2m) communications in virtualized cellular networks with mobile edge computing (mec). *IEEE Transactions on Mobile Computing*, 18(7):1541–1555, 2018.

[115] Yichao Chen, Kai Zhao, Heng Zhang, and Shibo He. A comprehensive physical layer security mechanism for mobile edge computing. *IEEE Internet of Things Journal*, 10(19):16816–16829, 2023.

[116] Liang Xiao, Xiaoyue Wan, Canhuang Dai, Xiaojiang Du, Xiang Chen, and Mohsen Guizani. Security in mobile edge caching with reinforcement learning. *IEEE Wireless Communications*, 25(3):116–122, 2018.

[117] Xiaojie Wang, Jiameng Li, Zhaolong Ning, Qingyang Song, Lei Guo, Song Guo, and Mohammad S Obaidat. Wireless powered mobile edge computing networks: A survey. *ACM Computing Surveys*, 55(13s):1–37, 2023.

[118] Yuncan Zhang, Weifa Liang, Zichuan Xu, and Xiaohua Jia. Mobility-aware service provisioning in edge computing via digital twin replica placements. *IEEE Transactions on Mobile Computing*, 23(12):11295–11311, 2024.

[119] Yi Li, Jian Shen, Pandi Vijayakumar, Chin-Feng Lai, Audithan Sivaraman, and Pradip Kumar Sharma. Next-generation consumer electronics data auditing scheme towards cloud-edge distributed and resilient machine learning. *IEEE Transactions on Consumer Electronics*, 70(1):2244–2256, 2024.

[120] Pasika Ranaweera, Anca Delia Jurcut, and Madhusanka Liyanage. Survey on multi-access edge computing security and privacy. *IEEE Communications Surveys & Tutorials*, 23(2):1078–1124, 2021.

[121] Hovav Shacham and Brent Waters. Compact proofs of retrievability. *Journal of Cryptology*, 26(3):442–483, 2013.

[122] Xinglin Zhang, Zhenjiang Li, Chang Lai, and Junna Zhang. Joint edge server placement and service placement in mobile-edge computing. *IEEE Internet of Things Journal*, 9(13):11261–11274, 2021.

[123] Yufang Gao, Zhibo Yan, Kanglian Zhao, Tomaso De Cola, and Wenfeng Li. Joint optimization of server and service selection in satellite-terrestrial integrated edge computing networks. *IEEE Transactions on Vehicular Technology*, 73(2):2740–2754, 2023.

[124] Muhammad Saqib Niaz and Gunter Saake. Merkle hash tree based techniques for data integrity of outsourced data. In *Proc. 27th GI-Workshop Grundlagen von Datenbanken*, Gommern, Germany, May 26-29, 2015.

[125] Eman Salim Ibrahim Harba. Secure data encryption through a combination of aes, rsa and hmac. *Engineering, Technology & Applied Science Research*, 7(4):1781–1785, 2017.

[126] Lingshuang Liu, Xiangman Li, Xue Qin, Jianbing Ni, and Xuemin Shen. Svtagg: Secure, verifiable, and traceable model aggregation for edge-assisted federated learning. *IEEE Transactions on Dependable and Secure Computing*, 2026.

[127] Xubin Wang, Zhiqing Tang, Jianxiong Guo, Tianhui Meng, Chenhao Wang, Tian Wang, and Weijia Jia. Empowering edge intelligence: A comprehensive survey on on-device ai models. *ACM Computing Surveys*, 57(9):1–39, 2025.

[128] Qiang Duan, Jun Huang, Shijing Hu, Ruijun Deng, Zhihui Lu, and Shui Yu. Combining federated learning and edge computing toward ubiquitous intelligence in 6g network: Challenges, recent advances, and future directions. *IEEE Communications Surveys & Tutorials*, 25(4):2892–2950, 2023.

[129] Jinglong Shen, Nan Cheng, Wenchao Xu, Haozhao Wang, Wei Quan, and Xuemin Shen. pfeddhpo: A differentiable approach for personalized hyperparameter optimization in federated learning. *IEEE Transactions on Cognitive Communications and Networking*, 12:412–426, 2026.

[130] Kaibin Wang, Qiang He, Feifei Chen, Hai Jin, and Yun Yang. Fededge: Accelerating edge-assisted federated learning. In *Proc. ACM Web Conference 2023*, pages 2895–2904, 2023.

[131] Peichun Li, Hanwen Zhang, Yuan Wu, Liping Qian, Rong Yu, Dusit Niyato, and Xuemin Shen. Filling the missing: Exploring generative ai for enhanced federated learning over heterogeneous mobile edge devices. *IEEE Transactions on Mobile Computing*, 23(10):10001–10015, 2024.

[132] Ngoc-Bao Nguyen, Keshigeyan Chandrasegaran, Milad Abdollahzadeh, and Ngai-Man Cheung. Re-thinking model inversion attacks against deep neural networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 16384–16393, 2023.

[133] Shokofeh Anari, Soroush Sadeghi, Ghazaal Sheikhi, Ramin Ranjbarzadeh, and Malika Bendechache. Explainable attention based breast tumor segmentation using a combination of unet, resnet, densenet, and efficientnet models. *Scientific Reports*, 15(1):1027, 2025.

[134] Mohamad Mansouri, Melek Önen, Wafa Ben Jaballah, and Mauro Conti. Sok: Secure aggregation based on cryptographic schemes for federated learning. *Proceedings on Privacy Enhancing Technologies*, 2023.

[135] Songge Zhang, Wen Wu, Lingyang Song, and Xuemin Shen. Efficient model training in edge networks with hierarchical split learning. *IEEE Transactions on Mobile Computing*, 24(10):10214–10229, 2025.

[136] Yumi Sakemi, Tetsutaro Kobayashi, Tsunekazu Saito, and Riad S Wahby. Pairing-friendly curves. *IETF Secretariat*, 2020.