MONASH University

Privacy-Preserving Data Exchange: Cryptographic Primitives and Solutions

Yanjun Shen

Doctor of Philosophy

A Thesis Submitted for the Degree of Doctor of Philosophy at
Monash University in 2024
Clayton School of Information Technology

Copyright notice

Abstract

The protection of privacy during data exchange is a critical concern in the digital age, encompassing a wide range of applications from personal confidential information exchange to sensitive business collaborative operations. This thesis explores privacy-preserving technologies for secure file and data sharing, synthesising studies across various real-life scenarios.

This thesis first investigates into a privacy-preserving file sharing web service designed for personal and business use. We present OblivSend, which effectively protects sensitive metadata while ensuring end-to-end encryption and user-controlled expiration by utilising lightweight cryptographic primitives. A prototype implemented on Hyperledger Fabric demonstrates OblivSend's efficiency and user-friendliness.

Further, we consider an advanced privacy-preserving file sharing scheme OblivShare that employs Oblivious RAM (ORAM) to enhance user control over file expiration, concealing expiration metadata from the server and maintaining full obliviousness regarding file access patterns and expiration states. This scheme ensures metadata privacy with poly-logarithmic complexity relative to the number of files, bridging a significant gap in the existing literature on secure file sharing.

In addition, we explore the application of privacy-preserving techniques in collaborative autonomous unmanned aerial vehicles (UAVs) operations with SecuPath, a framework integrating Secure Multi-Party Computation (SMPC) with generic path planning algorithms. SecuPath enables multiple entities to collaboratively compute optimal UAV paths without revealing sensitive and private data, elevating privacy and security standards in UAV operations while imposing minimal communication and computation overhead.

Collectively, the studies contribute to the growing body of research on privacy-preserving data sharing and collaboration technologies, offering robust solutions for secure data exchange in collaborative operations. The integration of advanced cryptographic protocols and privacy-preserving techniques ensures the confidentiality of sensitive data, fostering trust and collaboration in diverse domains.

Thesis including published works declaration

I hereby declare that this thesis contains no material which has been accepted for the award of any other degree or diploma at any university or equivalent institution and that, to the best of my knowledge and belief, this thesis contains no material previously published or written by another person, except where due reference is made in the text of the thesis.

This thesis includes *3* original papers published in conferences. The core theme of the thesis is *design and implement privacy-preserving data sharing and collaboration frameworks.* The ideas, development and writing up of all the papers in the thesis were the principal responsibility of myself, the student, working within the *Clayton School of Information Technology* under the supervision of *Prof. Joseph Liu, Dr. Xingliang Yuan, and Dr. Shifeng Sun.*

In the case of chapter *3, 4, and 5* my contribution to the work involved the following:

Thesis Chapter	Publication Title	Status	Nature and % of student contribution	Co-author name(s) Nature and % of Co-author's contribution	Co-author(s), Monash student
3	OblivSend: Secure and Ephemeral File Sharing Services with Oblivious Expiration Control	Published	60%. conception, system design, implementation, experiment, security analysis, writing the manuscript	1) Bin Yu. 10% domain knowledge, implementation 2) Shangqi Lai. 2%. domain knowledge, revising the manuscript 3) Xingliang Yuan. 10%. conception, design, revising the manuscript 4) Shi-Feng Sun. 10%. conception, security analysis, revising the manuscript 5) Joseph. K. Liu. 8%. conception, revising the manuscript 6) Surya Nepal. 2%. revising the manuscript	1) No 2) No 3) No 4) No 5) No 6) No

4	OblivShare: Towards Privacy-Preserving File Sharing with Oblivious Expiration Control	Published	60%. conception, system design, evaluation, security analysis, writing the manuscript	1) Xingliang Yuan. 15%. conception, domain knowledge, security analysis, revising the manuscript 2) Shi-Feng Sun. 10%. conception, security analysis, revising the manuscript 3) Joseph. K. Liu. 10%. conception, revising the manuscript 4) Surya Nepal. 5%. revising the manuscript	1) No 2) No 3) No 4) No
5	SecuPath: A Secure and Privacy-Preserving Multiparty Path Planning Framework in UAV Applications	In press	70%. conception, system design, security analysis, evaluation, writing the manuscript	1) Joseph. K. Liu. 15%. conception, domain knowledge, revising the manuscript 2) Xingliang Yuan. 12%. conception, domain knowledge, revising the manuscript 3) Shi-Feng Sun. 2%. conception, supervision 4) Hui Cui. 1%. supervision	1) No 2) No 3) No 4) No

I have renumbered sections of submitted or published papers in order to generate a consistent presentation within the thesis.

Student name: Yanjun Shen

Student signature: Date:

I hereby certify that the above declaration correctly reflects the nature and extent of the student's and co-authors' contributions to this work. In instances where I am not the responsible author I have consulted with the responsible author to agree on the respective contributions of the authors.

Main Supervisor name: Joseph Liu

Main Supervisor signature: Date:

Publications during enrolment

1. Shen, Y., Yuan, X., Sun, S. F., Liu, J. K., and Nepal, S. (2021). OblivShare: Towards privacy-preserving file sharing with oblivious expiration control. In Information Security Practice and Experience: 16th International Conference, ISPEC 2021, Nanjing, China, December 17–19, 2021, Proceedings 16 (pp. 126-146). Springer International Publishing.

2. Shen, Y., Yu, B., Lai, S., Yuan, X., Sun, S. F., Liu, J. K., and Nepal, S. (2022, December). OblivSend: Secure and Ephemeral File Sharing Services with Oblivious Expiration Control. In International Conference on Information Security (pp. 269-289). Cham: Springer International Publishing.

3. Shen, Y., Liu, J. K., Yuan, X., Sun, S. F., and Cui, H. SecuPath: A Secure and Privacy-Preserving Multiparty Path Planning Framework in UAV Applications. In Proceedings of the 29th Australasian Conference on Information Security and Privacy· July 15-17, 2024. Sydney, Australia.

Acknowledgements

First and foremost, I would like to take this opportunity to express my sincere gratitude to my supervisors: Prof. Joseph Liu, Dr. Xingliang Yuan and Dr. Shi-Feng Sun for their unwavering support and guidance throughout my PhD journey. Being a candidate who started with limited knowledge about security and cryptography, their expertise and experience in the domain provided the much-needed insights to enlighten my overall research and individual papers. They generously shared their wisdom, offering constructive feedback on all aspects of academic skills. Despite the interruptions during COVID, they remained committed to keep my research on track.

During my pregnancy and parental leave, my supervisors showed immense care not only for my academic progress but also for my personal well-being. They diligently worked to find arrangements that suited my unique situation, for which I am profoundly grateful.

I would also like to thank my research peers, Bin Yu, Shangqi Lai, whose support in domain knowledge, implementation and experimentation was crucial. Their collaboration and camaraderie made this journey smoother.

In addition, my heartfelt thanks extend to my panel members Assoc. Prof. Aamir Cheema, Prof. Chunyang Chen, Dr. Shujie Cui and Dr. Hui Cui for their invaluable feedback during milestone reviews. Their insights significantly shaped my future research directions and provided strategies to tackle the challenges I faced throughout my candidature.

I appreciate CSIRO's Data61 for their generous scholarship and extensive resources. My Data61 supervisor Dr. Surya Nepal regularly reviewed my progress and provided critical feedback to ensure I achieved the desired outcome. Meanwhile I appreciate all the non-academic support from Dr. Marthie Grobler during the course of my enrolment.

My acknowledgement also goes to the graduation research officers from the Faculty of Information Technology, who assisted with my milestones, compulsory courses and other administrative activities given the complexities introduced by my extended leave.

Last but definitely not least, I'm eternally grateful to my family Yaling Zhou, Jiang Shen, Yang Li, Chenkai Li. As a student mother, my parents and partner gave their unconditioned dedication and support, from taking care of every detail of my lives to caring for our little one who needs constant attention and affection, allowing me to focus on my research. Their boundless love, and constant belief in me have been my greatest source of motivation and strength to overcome all difficulties and challenges, pursue this journey with determination and strive to make them proud!

Contents

List of Figures

List of Tables

List of Acronyms

ABE **A**ttribute-**B**ased **E**ncryption. 10, 12

CCPA **C**alifornia **C**onsumer **P**rivacy **A**ct. 14

DOS **D**enial-**O**f-service. 28, 50, 57

E2EE **E**nd-to-**E**nd **E**ncryption. 2, 3, 5, 6, 8, 10, 12, 14–18, 29, 37, 59, 81

FHE Fully Homomorphic Encryption. 3, 12

FL **F**ederated **L**earning. 9, 84, 85

GBF **G**arbled **B**loom **F**ilter. x, 5, 19–21, 24–26, 28, 29, 31, 33, 35–37, 81

GC **G**arbled **C**ircuit. 9

GCS **G**round **C**ontrol **S**tation. 65, 71, 80

HE Homomorphic Encryption. 9

IB **I**dentity-**B**ased. 65

IBE **I**dentity-**B**ased **E**ncryption. 65

IoT **I**nternet **o**f **T**hings. 1, 10, 12

ORAM **O**blivious **RAM**. ii, x, 6, 7, 9, 12, 38, 41–47, 50–58, 81

PGP **P**retty **G**ood **P**rivacy. 9

PIR **p**rivate **I**nformation **R**etrieval. 10, 12

S2PC **S**ecure **T**wo-**P**arty **C**omputation. x, 45–48, 50, 51, 53, 56, 57, 66, 69–71, 74, 76, 78

SMPC **S**ecure **M**ulti-**P**arty **C**omputation. ii, 6, 7, 9, 12, 59, 60, 63, 67, 70, 71, 73, 75, 76, 79, 80, 82–85

UAV **U**nmanned **A**erial **V**ehicles. ii, 2–4, 6, 7, 11, 59, 62–69, 80–83

ZKP **Z**ero-**K**nowledge **P**roof. 10, 12

Chapter 1

Introduction

1.1 Motivation

The digital landscape, where data exchange is ubiquitous, witnesses an astounding volume of activity: every single minute, 188,000,000 emails are sent, 511,200 tweets are posted, 55,140 Instagram photos are shared over the internet [2]. This rapid exchange is propelled further by advancements in technologies such as cloud computing, Internet of Things (IoT), and autonomous systems. As these technologies proliferate, so does the need for reliable privacy-preserving mechanisms. Safeguarding sensitive data has thus become crucial to mitigate risks and uphold privacy and security standards.

However, just in the recent decade, we have witnessed some biggest ever high-profile data breaches including [3], [4], [5], [6], and [7] that impacted billions of accounts, exposing personal information including but not limited to user identities, biometric data, financial information, social media details. Far-too-frequent data leakages and increasing privacy awareness bring up a growing group of privacy concerned users who demand safe and private services. Users, encompassing both individuals and organisations, are increasingly wary of potential privacy breaches, data misuse, unauthorised access to their confidential information, and the prolonged retention of data on servers that can become a significant vulnerability over time.

The consequences of privacy breaches and data leakage can be severe. For individuals, such breaches can lead to identity theft, financial loss, and personal safety risks. For organisations, the repercussions include reputational damage, legal penalties, and

significant financial loss [8]. For instance, data breaches in healthcare can result in the unauthorised access to sensitive medical records, potentially jeopardizing patient confidentiality and trust. In the corporate sector, leaked intellectual property can lead to competitive disadvantages and substantial economic losses. According to IBM, the global average cost of a data breach in 2023 was USD 4.45 million [9].

Despite the critical need for enhanced privacy and security, existing research in these domains has several limitations. While end-to-end encryption (E2EE) ensures data confidentiality during transmission, many privacy-preserving data sharing solutions either fail to adequately protect metadata [10–12], which in general is data about data that for instance includes the time of a data communication session, the identities of the communicating parties, the frequency of data accessed, etc.. This not only can reveal significant information about the data being shared , or overlook the issue of data residing on servers longer than necessary, leaving sensitive information vulnerable to interception and misuse.

This thesis is motivated by the urgent need to develop innovative solutions that address the concerns. The research targets two pivotal domain areas: privacy-preserving file sharing with expiration control that enforces a file to expire at user's control including by when, for whom, after how many visits, etc.; and secure Unmanned Aerial Vehicle UAV path planning in collaborative setting. These areas are particularly pertinent due to the extremely sensitive nature of the data involved and the severe consequences of potential privacy breaches. By exploring advanced cryptographic primitives and developing innovative cryptographic frameworks, this thesis aims to enhance data privacy and security within these contexts, thereby contributing substantively to the broader field of secure digital communication and data exchange.

1.2 Research Question

The primary research question guiding this thesis is:

How can we design a practical privacy-preserving mechanism to enhance privacy and security while enabling data exchange?

In particular, developing and applying advanced cryptographic techniques to enable privacy-preserving data exchange efficiently, which includes file transfer and sharing, general data sharing and database connection.

To address this question, the following key sub-questions are considered:

- Ensuring secure data exchange is fundamental to protecting sensitive information and maintaining user trust in digital communication systems. A robust model needs to be developed that can cater to various application scenarios while ensuring data privacy and security. **How to design a proper model to enable secure data exchange in a privacy-preserving manner for various applications, including file sharing and UAV path planning?**

- Strong security and privacy are paramount in data exchange systems and applications, particularly when sensitive data is involved. While E2EE is commonly adopted to primarily secure data in transit, parameters used to secure data at rest or during computation lead to metadata privacy issues [13, 14]. It is essential to employ cryptographic methods that not only secure data but also ensure metadata privacy. **What cryptographic protocols can be employed to achieve strong security and enhance privacy while preserving system functionality in data exchange scenarios?**

- It is essential to ensure system efficiency to maintain usability, not sacrificing performance hence user experience. Existing solutions adopt cryptographic primitives such as Fully Homomorphic Encryption (FHE) that can be time-consuming [15]. To validate the applicability, effectiveness, and reliability of proposed cryptographic solutions, they must be deployed in practical scenarios and integrated with existing services to evaluate their performance and security under real-world conditions. **What are the performance implications of integrating advanced cryptographic protocols and how can the proposed privacy-preserving frameworks be optimised for scalability and real-world deployment?**

By addressing these research questions, this thesis aims to develop and validate advanced cryptographic primitives that enhance the privacy and security of data exchange. The

focus on designing robust models, achieving strong security and privacy, and ensuring practical applicability and integration is intended to safer and more reliable data exchange systems in various application domains.

1.3 Methodology

To address the research questions, this thesis employs a comprehensive approach combining theoretical analysis, cryptographic protocol design, and practical evaluation to address the research questions. The methodologies are structured as follows:

- **Literature Review and Analysis:** Conduct an extensive review of existing cryptographic techniques, privacy-preserving models, and secure data exchange frameworks. Analyze their strengths, limitations, and applicability to the research questions.
- **Cryptographic Protocol Design:** Develop novel cryptographic protocols that enhance privacy and security in data exchange, ensuring they can be applied to various scenarios such as file sharing and UAV path planning. Focus on achieving strong security and privacy while maintaining system efficiency.
- **Implementation and Prototyping:** Building proof-of-concept prototypes to demonstrate the feasibility and effectiveness of the proposed solutions.
- **Experimental Evaluation:** Conduct extensive testing and evaluations to assess the performance, security and efficiency of the implemented prototypes. Use both theoretical analysis and empirical experiments to validate the effectiveness of the proposed approaches.
- **Comparative Analysis:** Comparing the proposed solutions with existing approaches to highlight their advantages and potential improvements. Apply the developed protocols to practical scenarios to demonstrate the real-world applicability and benefits of the proposed solutions.

1.4 Contributions

This thesis aims to make contributions to the fields of cryptographic systems and privacy-preserving data exchange. The expected contributions are multifaceted, encompassing system design, the application of cryptographic primitives, and the development of practical systems and protocols. These contributions are tightly aligned with the primary research questions and sub-questions.

First, it presents a practical web system (OblivSend [16]) to support privacy-preserving file sharing with expiration control atop of E2EE. Further, the thesis introduces a framework that address the potential inference attack issue based on the proposed system and improves its privacy performance, demonstrating its usability in general file and data sharing scenarios. Finally, to extend the application of cryptographic framework in the privacy-preserving data exchange context for other real-world scenarios, the thesis also proposes a cryptographic scheme to support secure path planning in collaborative setting.

Remark. This thesis focuses on the confidentiality and privacy of data; however, does not guarantee the availability of data, which means data is available to the users at any time of the day, whenever and wherever required.

The following sections outline the system design, cryptographic primitives and application construct.

1.4.1 OblivSend: Secure and Ephemeral Privacy-Preserving File Sharing Service

Chapter 3 introduces OblivSend, a secure and ephemeral file sharing framework that provides users with advanced and oblivious expiration control to pre-set download constraints, such as limiting the number of downloads and the duration for which files are accessible. OblivSend supports comprehensive file expiration control and ensures expiration-metadata privacy, which can be integrated into other existing file-sharing web services. It addresses the challenge of how to achieve expiration control over protected expiration metadata by utilising cryptographic secret sharing schemes and Garbled Bloom Filters (GBFs). Additionally, it transfers the responsibility for checking

expiration conditions from the server to the clients, ensuring that users maintain control over their data. A proof-of-concept prototype of OblivSend was developed, including a smart contract program on Hyperledger Fabric, demonstrating negligible extra computation and communication overhead compared to traditional E2EE systems. This work directly addresses enhancing privacy and security in file sharing, specifically focusing on metadata protection and user control over data expiration. It also demonstrates real-world applicability and validates the theoretical contributions, addressing the overarching research question.

1.4.2 OblivShare: Privacy-Preserving File Sharing with Oblivious Expiration Control

Chapter 4 proposes OblivShare, a privacy-preserving file-sharing scheme based on Oblivious RAM (ORAM) to ensure the server is oblivious to file access patterns and expiration states, providing robust privacy guarantees without compromising performance. OblivShare hides user-defined download constraints from servers throughout the entire course of file upload, sharing, and download. By using synchronised tree-based ORAMs to store both file content and metadata, it ensures that servers cannot infer any information about file operations. Additionally, secure computation is employed for oblivious expiration control, guaranteeing that a single server cannot manipulate the expiration control result. A proof-of-concept prototype of OblivShare was developed and validated for complexity, performance, and security guarantees, demonstrating negligible extra computation and communication overhead on top of a primitive ORAM file sharing system. This work addresses metadata privacy and secure data management practices in file sharing.

1.4.3 SecuPath: Secure and Privacy-Preserving Multi-Party Path Planning Framework

Chapter 5 further studies other real-world application of privacy-preserving data exchange and introduces SecuPath, a novel approach to address privacy challenges in collaborative UAV path planning using secure multi-party secure computation (SMPC). This framework allows multiple entities to collaboratively plan UAV paths while preserving the privacy of sensitive input data, such as the drone's current location and planned

path, from exposure to each other or unauthorised parties. SecuPath incorporates cryptographic techniques to ensure the integrity and authenticity of the transmitted data, effectively protecting against prevalent risks in wireless communication environments. It balances efficiency and security by providing a detailed analysis of the communication and computation complexities introduced by the SMPC and cryptographic operations, demonstrating the protocol's practical feasibility. This work addresses securing UAV operations and collaborative computation, focusing on data privacy.

Overall, by addressing the limitations of current methodologies and applying advanced cryptographic techniques, this research advances the state-of-the-art in privacy-preserving systems. These contributions offer both theoretical insights and practical solutions that can be implemented in real-world scenarios.

1.5 Thesis Structure

This thesis is organised as follows. Chapter 2 reviews existing literature related to global background of this thesis and relevant cryptographic techniques. Related work specific to each contribution is given in their corresponding chapters. Chapter 3 introduces a privacy-preserving file sharing system OblivSend enhancing an existing web service that supports end-to-end encryption and user-controlled file expiration, meanwhile efficiently protects metadata. Chapter 4 presents an advance privacy-preserving file sharing scheme OblivShare based on Oblivious RAM (ORAM) and Secure Multi-Party Computation (SMPC) that ensures the server is oblivious to file access patterns and expiration states, which further improves the privacy performance of OblivSend. Chapter 5 extends the application to UAV path planning, and presents SecuPath, a secure framework for UAV path planning that leverages SMPC to enable collaborative computation of optimal paths without revealing private inputs. Chapter 7 further discusses some possible future directions in privacy-preserving systems, proposes potential future research directions, including scalability, advanced cryptographic techniques, and integration with emerging technologies. Finally, Chapter 6 summarises the key findings, contributions, and implications of the research, and outlines future work.

Chapter 2

Related Work

This chapter reviews the state-of-the-art literature in the field of privacy-preserving data exchange, including data sharing and collaborative computation. Our focus encompasses both theoretical advancements and practical implementations that address the increasing demand for robust privacy and security measures in various applications. This review is organised into three primary sections: privacy-preserving data sharing, secure multi-party computation (SMPC), and applications of privacy-preserving techniques in emerging technologies. Note that this chapter only discusses general background related to this thesis, and literature specific to each contribution is presented in chapter 3, 4, and 5 accordingly.

2.1 Privacy-Preserving Technologies

Privacy-preserving data sharing has become a critical area of research and various approaches have been proposed to ensure data confidentiality, integrity, and availability while sharing sensitive information.

2.1.1 End-to-End Encryption (E2EE)

One of the fundamental techniques in privacy-preserving data sharing is E2EE. E2EE ensures that data is encrypted on the sender's side and decrypted only by the intended recipient, preventing intermediaries from accessing the plaintext data. Pioneering works

such as Pretty Good Privacy (PGP) [17] and later, the Signal protocol [18, 19], have set the standard for secure communication.

2.1.2 Oblivious RAM (ORAM)

ORAM is a cryptographic primitive designed to hide access patterns to data stored in remote servers. It ensures that the server cannot learn which data items are being accessed, thus preserving the privacy of the data even in a scenario where the server is untrusted [20]. The Path ORAM scheme and its variants [21, 22] are notable examples that balance efficiency and security .

2.1.3 Secure Multi-Party Computation (SMPC)

SMPC enables multiple parties to jointly compute a function over their inputs while keeping those inputs private. This area has seen significant theoretical and practical advancements.

Foundational Protocols: The foundational work by Yao on garbled circuits and Goldreich, Micali, and Wigderson on general SMPC protocols laid the groundwork for secure computation [23]. These protocols allow for secure function evaluation where the parties learn only the output and nothing about each other's inputs.

Efficiency Improvements: Early SMPC protocols, while theoretically sound, were impractical due to high computational and communication costs. Subsequent research has focused on improving the efficiency of these protocols. Techniques such as homomorphic encryption (HE) [24], secret sharing, and optimised garbled circuits (GC) [25] have significantly reduced the overhead associated with SMPC.

Applications in Collaborative Environments: SMPC has found applications in various collaborative environments where data privacy is crucial. For instance, Federated Learning (FL) employs SMPC to train machine learning models across multiple data sources without compromising the privacy of individual datasets [26]. This technique is particularly relevant in scenarios like medical research and financial analysis, where data sensitivity is a critical concern.

2.1.4 Metadata Hiding Tools

While E2EE protects the content of the messages, metadata privacy remains a significant challenge. Metadata includes information about who is communicating with whom, when, and how often. Techniques such as Onion Routing, implemented in the Tor network, provide a degree of anonymity by routing communications through multiple nodes to obscure the source and destination of the data [12]. Recent advancements have focused on enhancing metadata privacy using cryptographic methods such as private information retrieval (PIR) [27] and differential privacy [28].

2.2 Applications in Emerging Technologies

The principles of privacy-preserving data exchange have been applied to various emerging technologies, enhancing security and privacy features across divers domains.

File Sharing: In the domain of file sharing, privacy-preserving techniques are increasingly important due to the sensitive nature of the personal or business data being exchanged. For instance, the adoption of Attribute-Based Encryption (ABE) in file sharing allows fine-grained access control and ensures that only authorised users can decrypt the shared files [29]. Another significant development is the implementation of blockchain-based file sharing systems that leverage decentralized storage and cryptographic techniques to ensure data integrity and privacy [30]. These technologies ensure that shared files remain confidential, even from the service providers, thereby addressing significant gaps in traditional file sharing services.

Internet of Things (IoT): IoT devices often collect and transmit sensitive data, necessitating robust privacy measures. Recent research has focused on integrating lightweight cryptographic protocols and secure data aggregation techniques to protect IoT data from unauthorized access and manipulation [31].

Blockchain and Distributed Ledgers: Blockchain technology inherently offers transparency and immutability; however, privacy remains a challenge. Techniques such as zero-knowledge proofs (ZKPs) and confidential transactions have been developed to enable private transactions on public blockchains such as Bitcoin and Ethereum. Projects

like Zcash and Monero are notable implementations that emphasise transaction privacy while maintaining the benefits of a distributed ledger [32, 33].

Autonomous Systems: The deployment of autonomous systems, such as UAVs and other autonomous vehicles, raises significant privacy and security concerns. Privacy-preserving path planning and secure communication protocols are critical to safeguard sensitive operational data from interception and manipulation, ensuring secure and reliable operations in dynamic environments [34].

Overall, the above applications of privacy-preserving techniques into these diverse domains demonstrates the versatility in addressing modern security and privacy challenges of various emerging technologies, which proves the necessity and importance of their integration into real-world applications.

Table 2.1 summarises existing top strategies in privacy-preserving data exchange, highlighting their advantages and limitations.

The field of privacy-preserving data exchange has made considerable strides in addressing the challenges associated with data privacy and security. From foundational cryptographic protocols to advanced applications in emerging technologies, the research landscape is rich with innovative solutions that enhance both security guarantees and performance. As the demand for secure and private data exchange continues to grow, ongoing research will undoubtedly yield further advancements, solidifying the role of privacy-preserving techniques in safeguarding sensitive information.

Technology	Applications	Advantages	Limitations
E2EE [18, 19, 35]	Messaging Apps, File Sharing, Email	Privacy guarantees	Does not protect metadata, key management challenges
FHE [24]	Secure Data Processing, Cloud Computation	Computation on encrypted data; privacy guarantees	Computational overhead, resource intensive, limited in collaborative settings
ABE [29, 36, 37]	Data Sharing, Cloud Storage	Granular access control	Computational overhead, key and policy management challenges
SMPC [25]	Data Analysis, Autonomous Systems, Machine Learning	Privacy guarantees, security against malicious servers, secure collaboration	Computational and communication overhead, scalability challenges
Differential Privacy [28, 38]	Data Analysis, Machine Learning	Privacy guarantees	Noise addition affects accuracy
ZKPs [32, 39]	Blockchain, Authentication, Data Sharing	Privacy guarantees, verifies information without revealing it	Computational overhead, implementation complexity
ORAM-Based Solutions [21, 22]	Cloud Storage, Database Queries	Access pattern privacy, metadata protection	Computational and communication overhead, scalability challenges
PIR [27, 40, 41]	Database Queries	Privacy in database access	Computational and communication overhead, limited in collaborative settings
Blockchain-Based Solutions [30, 42]	IoT, Data Sharing	Immutability, decentralized trust	Resource-intensive, scalability challenges

TABLE 2.1: Summary of Existing Privacy-Preserving Technologies and Their Applications.

Chapter 3

Secure and Ephemeral Privacy-Preserving File Sharing Service

Users have personal or business need to share most private and confidential documents; however, often at the expense of privacy and security. A sought after feature in the trending ephemeral context is to set download constraints of a particular file - a file can only be downloaded a limited number of times and/or for a limited period of time. Emerging end-to-end encrypted file sharing services with enhanced expiration control are attempts to meet the needs. Although such new services have drawn much attention, their server can still observe and control metadata of such download constraints, which could reveal partial data information. To address this challenge, in this chapter, we propose OblivSend, a privacy-preserving file sharing web service that 1) supports end-to-end encryption, 2) allows a limited period of time and a limited number of downloads at users' control, and 3) protects expiration control metadata from the server efficiently by lightweight cryptographic primitives. We develop a proof of concept prototype implemented in Hyperledger Fabric on a Research Cloud and evaluations demonstrate that our prototype can function as intended to achieve privacy of metadata without sacrificing user experience.

3.1 Introduction

Individuals' security and privacy are fundamental rights, not only in the real world but also the digital universe. Far-too-frequent high-profile data leakages and mass surveillance projects [3, 5] remind people of the vulnerability and sensitivity of personal data, hence the increasing privacy awareness brings up a growing group of privacy concerned users, who demand safe and private services including file sharing services. Existing regulations and acts to protect personal data, such as such as the General Data Protection Regulation (GDPR) [43] and the California Consumer Privacy Act (CCPA) [44], also impose on service providers to grant individuals control over their private information. There are a quantity of file sharing services in the market, such as *Gmail*, *DropBox*, even instant messaging tools like *WhatsApp*, but these services often have constraints: emails and instant messengers have harsh attachment size limit that a 30-second 1080p footage can easily exceed; and most file hosting services do not support client-side encryption or self-destruct, which means your long-lived files can be decrypted any time simply by a rogue employee or if cooperated with government surveillance.

Among existing services, *Firefox Send* [1], officially launched by Mozilla in March 2019, allows to share files up to 2.5GB at a time with end-to-end encryption (E2EE) from any modern browsers. *Send* empowers a user to encrypt a file and its metadata including file name, size and type before uploading (see figure 3.1); then send the file link (via a secure channel of the user's choice to dedicated receiver(s) who can then request the service for downloading. In addition, it offers extra security control to users over the files they share: setting files to expire after a certain period of time or number of downloads. *Send* incorporates two most desired features, **E2EE** and **ephemeral**, which meets personal needs of more secure connections and intimate sharing.

However, limitations of *Send* are also apparent: 1) Users have to fully trust the service to honestly check if a file has expired. 2) Users send expiration control metadata, i.e. download number and time limits that are used to check if a file has expired on a download request, to *Send* in plaintext, which can be used to indicate the popularity and sensitivity of specific file(s).

Metadata privacy has drawn increasingly attention after the Edward Snowden leaks. "If you have enough metadata, you don't really need content", "we kill people based

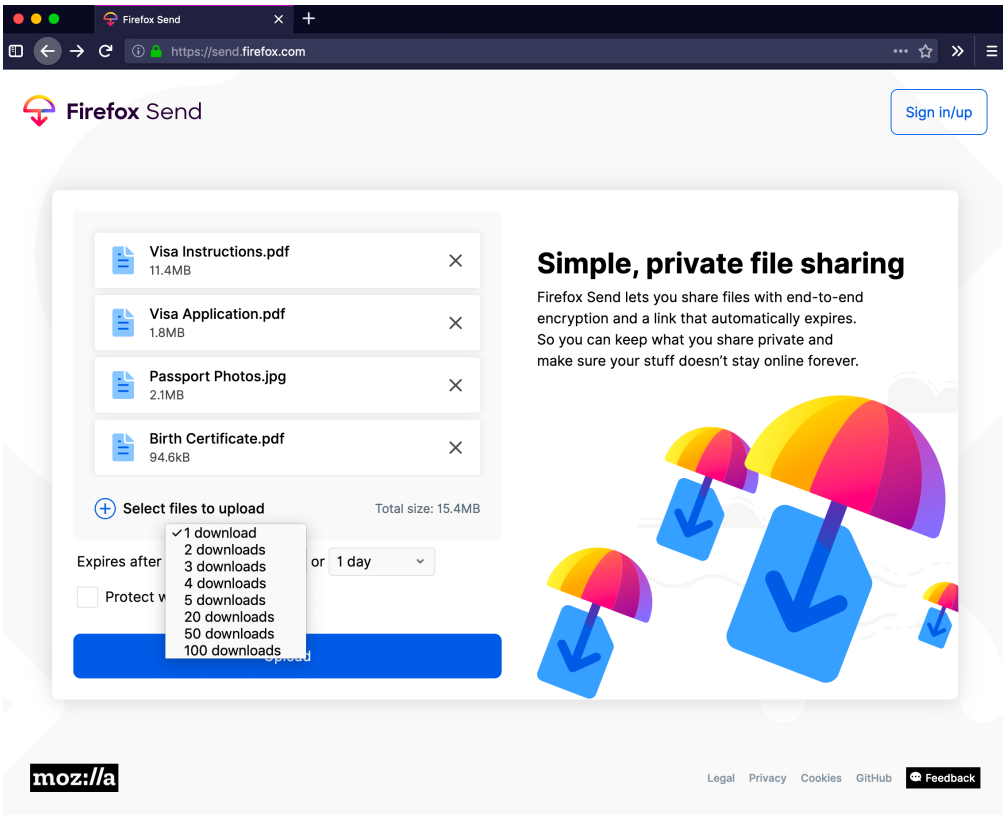


FIGURE 3.1: Firefox Send. Choose when a file expires and the number of downloads.[1]

on metadata" [45, 46]. Since sharing a file is similar to calling and messaging someone, metadata access in file sharing is also concerning.

3.1.1 Motivation

To illustrate the motivation of hiding the expiration control metadata (expiration metadata for short in the rest of the chapter), we present some privacy issues resulted from potential leakage of expiration metadata, even with E2EE file sharing systems.

User story 1: sensitivity derived from expiration metadata. Alice is an oncologist, and due to COVID-19, she shares files with patients and other contacts in an E2EE system. Alice shares electronic medical records with her patients and sets each to expire after 1 download or 1 day, and general files without expiration conditions. With the knowledge of the expiration metadata, a curious server knows that Alice shares some files of strict access, hence deduces they are sensitive. Bob is a patient of Alice and downloads his report from the system. With Alice's identity and the sensitivity of the

file, the server thus infers Bob is suspected to have cancer, which has implications on his insurance and other indemnities.

User story 2: frequency derived from expiration metadata. A medical institution distributes confidential reports to different teams of professionals using an E2EE system and sets the download number limit as the number of team members. For instance, a cancer service team has 6 doctors and all files shared to the team are set to expire after 6 downloads. With side information about the institution's teams, likely available via a web search from its homepage, a server can infer the disease in each file. Bob visits Alice in the institution, even without Alice's identity, a server can reasonably deduce Bob's disease from the file Alice downloads without decryption.

User-input expiration metadata by itself may not be damaging; however, when combined with other metadata, scaling to a great population, and observed in aggregate, it can be meaningful and reveal sensitive information [46, 47].

3.1.2 Our Contributions

We propose OblivSend, a secure and ephemeral file-sharing system that for the first time provides users with advanced and oblivious expiration control. OblivSend puts forward a new framework of a file-sharing service that not only supports comprehensive file expiration control, but is also expiration-metadata-private, which is a generic solution that can be integrated into other file sharing services. To understand our contribution, we now outline the main challenges OblivSend aims to address.

Challenge 1: how to achieve expiration control over protected expiration metadata? We define expiration metadata as a download number limit and a download time limit to ensure expiration control of a file sharing service. Users are not able to download a file upon exceeding the pre-set number of downloads or elapse time. Unfortunately, though more secure file sharing services promise expiration control as a premium feature [1, 11, 14], inadequate discussion has since occurred to understand the privacy of expiration metadata and how they impact the way people experience file sharing over the network. To the best of our knowledge, whereas many scholars focus on protection of general security control metadata in file sharing such as user identity and

access pattern [48], there is no prior research aiming to prevent leakage of expiration metadata, hence a gap exists to address such expiration-metadata-privacy.

Challenge 2: how to grant expiration control power to a user rather than a service provider? Users pursue E2EE file sharing services because they do NOT trust a service provider to keep their encryption keys. Therefore, it is an obvious defect that users adopt an E2EE service, but completely trust the service to check the expiration conditions and control if a file can be downloaded and decrypted. We assume a server that provides file storage services and fulfils upload and download requests; but wants to learn the expiration metadata, furthermore, actively manipulate its internal download state that is compared to expiration conditions.

OblivSend supports E2EE meanwhile protects the expiration metadata through the entire course with oblivious expiration control. Overall, our contributions are:

1 **Hiding expiration metadata.** We hide expiration metadata of file sharing services, both download number and time limits, through the entire life cycle by adopting cryptographic secret sharing scheme and garbled Bloom filter.

2 **Oblivious expiration control.** We enforce oblivious expiration control by transferring the responsibility for checking expiration conditions from a server to clients.

3 **Precautionary detection.** We audit and precautionarily detect forged download state at a serve side leveraging the immutability of smart contract.

4 **Implementation.** We implement a OblivSend prototype and develop a smart contract program cover a Hyperledger Fabric network. We also evaluate the performance, which approves that OblivSend has negligible extra computation and communication overhead on top of a traditional E2EE file sharing system.

3.2 Related Work

3.2.1 Existing Secure File Sharing Services

Table 3.1 compares several existing secure file sharing applications or web services. We organise the comparison by the following properties: 1) Does the service support E2EE?

Product	E2EE	Time Lim.	No. Lim.	Oblivious
OblivSend	✓	✓	✓	✓
Firefox Send [1]	✓	✓	✓	✗
DropSecure [11]	✓	✓	Future	✗
SendSafely [14]	✓	✓	✗	✗
WhatsApp [49]	✓	Future	✗	✗
Digify [50]	✗	✓	✗	✗
Dropbox [51]	✗	✗	✗	✗

TABLE 3.1: Secure File Sharing Services.

2) Does the service support user-controlled file expiration? 3) Is the server oblivious of the expiration control?

While E2EE has become increasingly preferred when users choose a web service, additional impermanence or ephemeral feature, such as *Telegram Secret Chat* [52] and *Gmail confidential mode* [53], inspired and pioneered by *SnapChat* is another highly pursued trend that dominates the internet [1, 11, 14, 49]. With certain expiration control, people can be confident that what they share is only accessible to dedicated users for a limited period of time or number of times, and they are able to wipe out all their "secrets" with a Thanos snap so that nothing will stay in a server for longer than necessary and become a vulnerability later. Emerging file sharing services that grant users expiration control are the file-sharing versions of *Snapchat*.

In addition to *Send*, *DropSecure* and *SendSafely* claim to offer zero-knowledge E2EE. *DropSecure* (premium) automatically destroys files from the servers after seven days and plans to support download number limit in the future. As an instant messaging service, *WhatsApp* also supports E2EE attachments, and has been developing its "Expiring Messages" feature. Services such as *Dropbox* and *Digify* provide an addition layer of security on top of server-side encryption, which enables a user to double encrypt files or folders by setting a password.

Some email services also support E2EE but the size of attachments is limited to 25MB per email [54, 55], and other approaches to share file securely online have low usability for none technical users [35, 56, 57].

3.2.2 Privacy-Preserving Web Services

In recent literature, researchers and practitioners have made great efforts to propose and promote privacy-preserving systems. [58–60] allow privacy-preserving data aggregation that enables monitoring, collection and analysis of statistics on the population without explicitly learning each user's individual contribution. [58, 61, 62] offers privacy-preserving safe browsing experience, while the former two protect sensitive user information entered into a browser and cached from auto-filled forms from sophisticated attackers and malware without trusting any part of the web applications, and the latter detects and blocks unsafe websites without leaking either users' browsing history or the lists of unsafe URLs maintained by third-party blacklist providers. [63] enables privacy-preserving smart contracts that address blockchain's lack of confidentiality by separating consensus from execution. Similar attempts have been made to develop metadata-private systems, but with a limited focus on secure messaging, either group messaging [64, 65], or private messaging [66–69], and private presence and notification [70].

OblivSend is aiming to strike a good balance of the above desired features and usability; further, address the security challenges mentioned in § 3.1.

3.3 Preliminaries

OblivSend makes black box use of secret sharing, garbled Bloom filter (GBF), smart contract, collision-resistant pseudorandom functions and hash functions.

Notation. We define parameters, entities, denotations in OblivSend in Table 3.2.

Secret Sharing [71] is a fundamental cryptographic primitive that splits a secret s into n shares such that the secret s can be recovered efficiently with any subset of t or more shares. With any subset of less than t shares, the secret is unrecoverable and the shares give no information about the secret. A scenario when $t = n$ is applied in this scheme, and a secret related to a file encryption key can be restored via simple $\oplus(XOR)$ operations.

The scheme generates $n-1$ random bit strings $r_1, r_2, ..., r_{n-1}$ of the same length as the secret s, and computing $r_n = r_1 \oplus r_2 \oplus, ..., \oplus r_{n-1} \oplus s$. Each r_i is a share of the secret s. It is apparent that s can be recovered by (XOR)ing all the shares $r_1 \oplus r_2 \oplus, ..., \oplus r_{n-1} \oplus r_n$

Notation	Description
λ, σ, l	security parameters
N (T)	a finite set of integers from 1 to download limit n (or elapsed time unit limit t) that denotes all download *numbers* (or *time units*) allowed by OblivSend
L_N (L_T)	a finite set of integers from 1 to l_N (or l_T) where l_N (l_T) is the download *number* (or *time*) limit
S_U	a user secret shared securely between a data owner and clients
E_N (E_T)	a key element bound to download *number* (or *time*)
FK	a secret key used for file encryption that is composed of E_N and E_T
K_N (K_T)	a finite set of protection keys enerated from a pseudorandom function on S_U to cipher a *number* (or *time*) key element
C_{E_N} (C_{E_T})	a finite set of cipher *number* (or *time*) elements encrypted using each element in K_N (or K_T)
AB_N (AB_T)	a finite set of encrypted payload messages used for GBF_N (or GBF_T)
M_N (M_T)	a map of all elements in N (or T) paired with their hash values
k	number of hash functions in a garbled Bloom filter
m	length of a garbled Bloom filter
H	a set of k independent hash functions $\{h_0, ..., h_{k-1}\}$ each $h_i(j)$ on input j outputs an index number over $[0, m-1]$ uniformly
GBF_N (GBF_T)	a garbled Bloom filter encoding a set N (or T)
PRF	a pseudorandom function $\{0,1\}^{2\sigma} \xleftarrow{\$} \{0,1\}^{\sigma} \times \{0,1\}^{*}$ that on input a σ-bit key and some string, outputs a 2σ-bit pseudorandom string

TABLE 3.2: OblivSend Notation

and any subset of less than n shares reveals no information about the secret and the secret is unrecoverable.

Garbled Bloom Filter (GBF) is a data structure introduced in [72] that encodes a set of at most n λ-bit strings in an array of length m, which supports membership query with no false negative and negligible false positive. Instead of generating an array of bits of 0 and 1 in a standard Bloom filter [73], a GBF adds secrets of $x \in S$ using the XOR-based secret sharing scheme depicted above. While querying the GBF for membership checking, only if $y \in S$, the XOR operations will recover y from the GBF .

To add an element $x \in S$ to a garbled Bloom filter, the element x is first split into

k λ-bit strings using the the XOR-based secret sharing scheme depicted in the previous paragraph, then is mapped by k hash functions into k index numbers and each location $h_i(x)$ is allocated with one secret share to form an array of λ-bit strings. To query an element y, the element y is mapped by the same k hash functions into k index numbers and all bit strings at the corresponding array locations $h_i(y)$ are collected and XORed together. To query an element y, the element y is mapped by the same k hash functions into k index numbers and all bit strings at the corresponding array locations $h_i(y)$ are collected and XORed together. If $y \in S$, the XOR operation will recover y as a result of XORing its k shares retrieved from the garbled Bloom filter by their indices. If $y \notin S$, then the probability of the XOR result is the same as y is negligible in λ.

Remark: During the course, a specific location $j \leftarrow h_i(x)$ may have been occupied by a share of a previously added secret, and in this case the existing share stored at $GBF[j]$ is reused (line 18 in Algorithm 2), otherwise, the previously added secret will not be recoverable in the query phase. For instance, in Figure 3.2, when we add s_2, $GBF[7]$ has already been occupied by s_1^3 and reused as a share of s_2, since $s_2^3 = s_2^1 \oplus s_1^3 \oplus s_2$, $s_2 = s_2^1 \oplus s_1^3 \oplus s_2^3$ still stands.

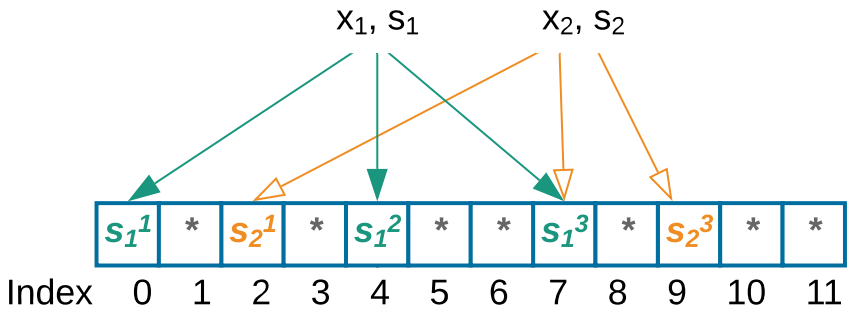


FIGURE 3.2: Adding Secrets into a GBF. x_1 is hashed into 3 numbers 0, 4, 7, and 3 shares of s_1 are allocated to index 0, 4, 7; x_2 is mapped to 2, 7, 9, where an existing share s_1^3 is reused at index 7.

According to [72], the false positive probability of a garbled Bloom filter, which is the probability that for $y \notin S$, the recovered string from XORing all $GBF_S[h_i(y)]$ is the same as y coincidentally, is at most $2^{-\lambda}$.

In our proposed framework, x is either a download number or time unit, and the secret whose shares are added is either a cipher key element or a payload message. When y, which is the current download count or elapse time unit, is within the corresponding download limit, the cipher key element can be recovered.

Smart Contract is a "computerised transaction protocol" first coined by Nick Szabo in 1997 [74] that digitally enforces secure relationships and credible transactions over public networks via a blockchain. It intends to minimises trust and eliminates needs for third parties, and common application of a smart contract ranges from financial to logistics, healthcare to energy resources. [75–77].

Since blockchain is a massive decentralised ledger by nature, once a transaction is validated and written to the blockchain, it can neither be deleted nor modified without a majority of collusion, which makes the blockchain immutable. Our proposed scheme builds smart contract into the framework because it is automatic and direct, fast and cheap, and tamper-proof.

3.4 System Overview

In OblivSend, when a data owner uploads a file, the data owner is able to set the file to expire after a number of downloads or period of time. When a client makes a download request to OblivSend, OblivSend sends the encrypted file to the client, and the client can only decrypt the file if it has not expired. To understand how OblivSend performs these operations securely, we present an overview of OblivSend's design, threats and security goals.

3.4.1 Intuition

We now present several attempts, beginning with naive approaches that are obviously insecure and proceeding to a practical version of our proposed scheme.

Attempt #1: Client-end encryption of expiration metadata. A natural approach is to simply encrypt the expiration metadata and send the resulting ciphertext to a server. Assuming we use proper encryption schemes so that the server is able to verify a client's download request by comparing its current internal download state with

the ciphertext of download limits. Based on the comparison results, the server determines to send the encrypted file to the client or not. This design should offer metadata privacy even when they are held by a malicious server.

It is obvious that although it prevents tampering with the stored metadata, it does not prevent a malicious server from manipulating the comparison result. Such an attacker can allow downloads after a file has expired by using an old internal state, and vice versa. Further, the server is able to deduce the expiration metadata based on its internal download state on the boundary if a new download request fails the comparison.

Attempt #2: Server is oblivious of the comparison. Instead of comparing the its internal download state with encrypted download limits, a server computes an output using its internal download state, and sends the output and the encrypted file to the client. Only if the internal download state fulfils the pre-set expiration conditions, the output can unveil the file encryption key.

This approach deprives the server of the ultimate power to grant or deny a download request. The server is oblivious of the comparison outcome, moreover not able to learn about the expiration metadata. However, it still requires a single trust party, the server, to honestly compute the output using the correct internal download state. It is apparently not desirable because a single authority is easy to attack and collude. Besides, this protocol does not prevent the server from replaying an old computation result to the client. In practice, such an attacker can always permit a download request regardless of the pre-set expiration conditions.

Attempt #3: Use smart contract to audit state. In order to address the single trust issue, we require decentralised trust that reduces the trust level of a single server. We use a public ledger on a blockchain to keep a world download state, and a client interacts with a smart contract to post and get the world download state. The server still computes an output but the output is encrypted using its internal download state. The client requests the world download state from the smart contract, and only if the world download state in the ledger is consistent with the internal state used by the server, the output can be recovered. It is impossible for an adversary to tamper the world download state in the ledger without a majority of collusion because of the immutable nature of a blockchain. This is a practical and more accepting setting, and exposes no extra trace on the blockchain by publishing hashed digests rather than real data.

Remark. Like the previous attempt, the protocol described above does not prevent a server from replaying old computation results. However, it precautionarily detects such replaying attack, and indeed prevents a client from downloading an expired file even if the server keeps replaying old computation results, and hopefully ensures that the server gains nothing from such attacks.

Our design intuition. The above attempts provide an intuition for our ideas, and OblivSend uses the following techniques to realise our intuition. OblivSend hides expiration metadata leveraging secret sharing and GBF. It first maps the download count and time unit up to system default download limit (i.e. the maximum download count or time unit allowed by OblivSend) into a GBF's locations, and puts secret shares of a cipher key element in locations mapped from counts or time units under the download limit but a payload message in those over the limit. OblivSend builds two GBFs for number and time limits respectively. On a download request, an OblivSend's server computes both GBFs using its internal download state and sends the result to a client. If its internal download state does not exceed either limit, shares from corresponding locations of each GBF result recover the corresponding cipher key element.

Further, OblivSend takes advantage of the immutability of a blockchain to audit the server integrity. The client requests for the world download state on a blockchain via a smart contract, and validates the server's internal state. Only if the states are consistent can a client decrypt the cipher key element, and with both key elements, the client can rebuild the file decryption key and decrypt the file. The server is oblivious of all the subsequent validation and decryption operations after handing over the GBF computation results, hence is no longer a central authority to grand or deny a download request and never knows if the expiration conditions have reached or not. Note this is a loose description, the detailed construction is elaborated in § 3.5.

3.4.2 Framework

Figure 3.3 shows that OblivSend consists of four parties and three phases:

- **Data Owner** is a sender of file(s), who generates file encryption keys, encrypts the file(s), sets download limits, and encodes the file encryption key and expiration metadata into GBFs before uploading them to a server.

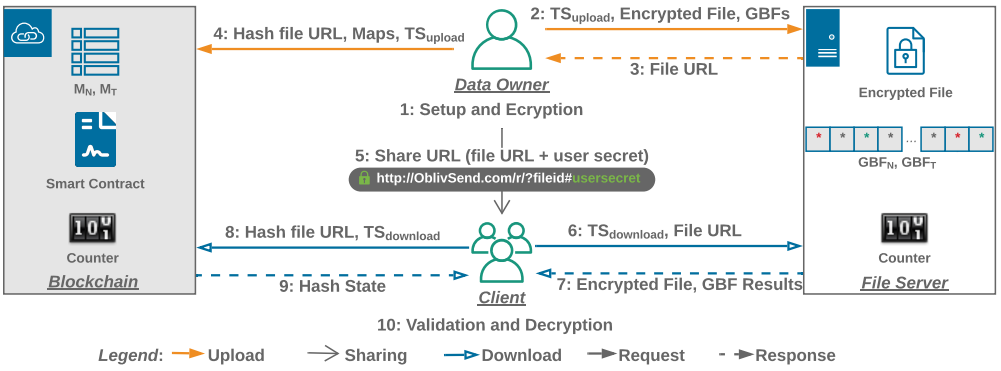


FIGURE 3.3: High-level Framework. Shaded areas are introduced by OblivSend.

- **Server** is where the encrypted files and GBFs are stored, who computes the GBFs using its internal download state and returns the GBF results and encrypted files on download requests.
- **Smart Contract** on a blockchain is a program that truthfully receives, updates, and returns the world download state.
- **Client** is an expected recipient of the file(s) who retrieves the encrypted file(s) and GBF results from the server, and the world download state from the smart contract. The client validates the GBF results before it can decrypt the file(s).

Upload consists of 4 steps corresponding to steps on Figure 3.3: 1) Data Owner generates a file encryption key FK composed of a *number* E_N and a *time* key element E_T, and encrypts a file; then populates hashes for each allowed download count and time unit and puts them into two maps; it generates a user secret S_U and ciphers the two key elements using the user secret and hashes, then put the cipher key elements C_{E_N} and C_{E_T} into two GBFs GBF_N and GBF_T based on the download limits l_N and l_T accordingly. 2) Data Owner uploads the GBFs, the encrypted file and an upload timestamp to Server. 3) Server returns a file URL. 4) Data Owner sends a hash value of the file URL as a unique identifier, two maps of hashes, and an upload timestamp to Smart Contract.

During **Sharing**, Data Owner appends a user secret S_U to the file URL and sends the share URL to trusted Client(s) via some out-of-band communication independent of OblivSend (as discussed in § 3.4.3).

Download includes the following operations shown on Figure 3.3: 6) Client renders the share URL, and requests Server for the encrypted file by the file URL and an download

timestamp. 7) Server returns the encrypted file and the GBF computation results using its internal download state. 8) Client queries Smart Contract for hashes of the world download state by the hash file URL and the download timestamp. 9) Smart Contract returns the hashes by looking up the world download count number and elapse time unit in the maps on the blockchain. 10) Client imports the user secret S_U from the share URL and validates Server's download state by decrypting the GBF results with the user secret S_U and hashes. If Server's internal state is consistent with the world state on the blockchain, Client can decipher the GBF results. If neither the current download count nor the elapse time unit has exceeded the pre-set download limits, the deciphered GBF results are the two key elements E_N and E_T, and Client can rebuild the file encryption key FK to decrypt the file.

OblivSend changes *Send*'s framework to a minimal extend while improving its security performance. A notable variation is the usage of GBF to hide expiration metadata and ensure server obliviousness. Another extension is the introduction of Smart Contract on the left hand side of Figure 3.3, which provides a mechanism to audit the server integrity to detect any forged download state. We make such minimal extension on purpose so as OblivSend's oblivious expiration control is generic and can be applied to other file sharing services.

3.4.3 Threat Models and Security Goals

Threat models. We assume that an attacker can compromise any set of users and an OblivSend server, while at least one client is honest. In particular, OblivSend considers the following threats:

a) The server would sniff the expiration metadata, hence deduces valuable information of encrypted data.

b) The server would forge its internal download state. A replay attack is typical that allows a client to download after a file has expired.

c) An attacker controlling a client would try to compromise the privacy of data that is not shared with him/her.

Assumptions. *Out-of-band communication.* A data owner in OblivSend shares a URL with client(s) through third party end-to-end secured channels of their own control, such as *Telegram* [52], *Signal* [13]. The share URL can be sent via secure messages similar to communicating user identities or notifications in [48, 78]. OblivSend only uses such out-of-band communication once at the sharing stage, which is the same as *Send* and a common practice of other secure file sharing systems [1, 51], keeping all other activities within OblivSend.

Blockchain. OblivSend makes black-box use of a blockchain and a standard assumption that a blockchain is immutable. If any set of peer nodes are compromised or an attacker seeks membership of a blockchain, who attempt to mutate the world download state on a blockchain, it becomes visible to all participants who, by a simple majority of votes [79], can prevent such unlawful actions from happening. We also inherent general blockchain security assumptions, which is not narrated in this chapter.

Anonymous Network. In order to hide other metadata during file sharing, OblivSend assumes the data owner and clients communicate with the server in an anonymous manner that does not reveal their network information via existing tools such as Tor [12] or secure messaging [64, 66, 69] based on decentralised trust.

Security goals. OblivSend sets the following goals to address the above threats:

a) **Expiration metadata privacy.** OblivSend ensures expiration metadata is totally at a data owner's control, and not visible in transit or at rest on either the server or the blockchain.

b) **Server integrity auditing.** OblivSend uses the public audit-ability of a blockchain underpinned by its immutability feature to detect forged download state on a server before actual decryption takes place.

c) **File confidentiality.** A client is not able to recover the encryption key and decrypt the file by guessing the current download state.

d) **General metadata protection.** OblivSend does not reveal to the blockchain the following in plaintext: user IP, file URL, download limits, download state, upload and download timestamps.

OblivSend achieves the goals above based on common cryptographic assumptions and we provide extended discussion in § 3.5.1.

OblivSend does not address denial-of-service (DOS) attacks.

3.5 Detailed Construction

In this section, we describe key components of OblivSend and present their algorithms. We also provide security analysis of OblivSend.

Upload.Generate. Run by the data owner at the beginning of the upload phase to generate a user secret S_U, a file encryption key FK that composed of two key elements (E_N, E_T), and two maps (M_N, M_T) of all members in the system allowed download number set N and time unit set T with their hashes. The data owner also generates a protection key from a pseudorandom function on S_U and a hash value of $n \in N$ or $t \in T$ (line 6, 12 in Algorithm 1) and ciphers the key elements using the protection keys (line 7, 13 in Algorithm 1). The data owner encrypts a file using FK, sends S_U to clients during the sharing phase, and sends M_N and M_T to a smart contract. The cipher elements (C_{E_N}, C_{E_T}) and payload messages (AB_N, AB_T) are to be added into GBFs in Upload.Encode.

Algorithm 1: Upload.Generate

Input: σ, N, T
Output: FK, S_U, C_{E_N}, C_{E_T}, AB_N, AB_T, M_N, M_T
1 Initialisation;
2 Generate $S_U, E_N, E_T \xleftarrow{\$} \{0,1\}^\sigma$;
3 Compute $FK \leftarrow hash(E_N||E_T)$;
4 **for** $p \in N$ **do**
5 $\quad \hat{p} \leftarrow hash(p)$, $M_N[p] \leftarrow \hat{p}$;
6 $\quad K_N[p] \leftarrow PRF_{S_U}(\hat{p})$;
$\quad$ // protect the key element
7 $\quad C_{E_N}[p] \leftarrow Enc_{K_N[p]}(E_N)$;
8 $\quad AB_N[p] \leftarrow Enc_{K_N[p]}(\text{“}ABORT\text{”})$;
9 **endfor**
10 **for** $q \in T$ **do**
11 $\quad \hat{q} \leftarrow hash(q)$, $M_T[q] \leftarrow \hat{q}$;
12 $\quad K_T[q] \leftarrow PRF_{S_U}(\hat{q})$;
13 $\quad C_{E_T}[q] \leftarrow Enc_{K_T[q]}(E_T)$;
14 $\quad AB_T[q] \leftarrow Enc_{K_T[q]}(\text{“}ABORT\text{”})$;
15 **endfor**

Upload.Encode. Run by the data owner to construct two GBFs (GBF_N, GBF_T) encoding the cipher key elements (C_{E_N}, C_{E_T}). It has two purposes: 1) to encode the elements that can compose the file encryption key into GBFs (line 17 in Algorithm 2) to ensure E2EE; 2) to map the download limits into GBF locations (line 4, 7 in Algorithm 2) to preserve expiration metadata privacy meanwhile enforce expiration control. The data owner sends GBF_N and GBF_T to a server together with the encrypted file C and an upload timestamp. Algorithm 2 takes the number element and download number limit as an example. For download time limit, it takes inputs T, L_T, m, k, λ, C_{E_T}, AB_T, H, and outputs GBF_T.

Download.Compute. Run by the server on a download request from a client to compute GBF_N and GBF_T using the current download count c and elapse time unit e. If $c \in L_N$ and $e \in L_T$, shares of the cipher key elements will recover $C_{E_N}[c]$ and $C_{E_T}[e]$, otherwise cipher payload messages ($AB_N[c]$, $AB_T[e]$). The server returns the computation results R_N, R_T to the client.

Download.Validate. Run by the client to decipher the GBF results R_N and R_T using the hashes of current download state ($\hat{c^*}, \hat{e^*}$) fetched from the smart contract. Client first regenerates the protection keys $K_N[c^*]$ and $K_T[c^*]$ based on S_U, then decipher the GBF results. If both elements are deciphered successfully (line 4, 5 in Algorithm 4), it proves that the current download state c and e used by the server are authentic, i.e. $c = c^*$ and $e = e^*$; otherwise, we must have $K_N[c^*] \neq K_N[c]$ and/or $K_T[e^*] \neq K_T[e]$ that are used to envelop the key elements in Upload.Generate (see line 6, 12 in Algorithm 1). The unveiled secrets ($E_N{}^*, E_T{}^*$) are either (E_N, E_T) or "ABORT" depending on whether the current download state satisfies the download limits. Only if $c \in L_N$ and $e \in L_T$, the secrets are the key elements and the client can reassemble FK and eventually decrypt the file.

3.5.1 Security Guarantee

We now present security guarantees of OblivSend with respect to the goals given in § 3.4.3.

Expiration metadata privacy. OblivSend hides expiration metadata from a server yet is able to enforce the expiration control by using GBFs in Upload.Encode. The

Algorithm 2: Upload.Encode

Input: N, l_N, m, k, λ, C_{E_N}, AB_N, H
Output: GBF_N

Initialise GBF_N;
for $p \in N$ **do**
  $emptySlot \leftarrow -1$;
  // under download limit
  if $p \leq l_N$;
    $final \leftarrow C_{E_N}[p]$;
    for $i = 0$ **to** $k - 1$ **do**
      // get an index via hashing
      $j \leftarrow h_i(p)$;
      if $GBF_N[j] ==$ NULL **then**
        if emptySlot == -1 **then**
          // Reserve a location
          $emptySlot \leftarrow j$;
        else
          // generate a secret share
          $GBF_N[j] \xleftarrow{\$} (0,1)^\lambda$;
          $final \leftarrow final \oplus GBF_N[j]$;
      else
        // reuse an existing share
        $final \leftarrow final \oplus GBF_N[j]$;
    endfor
    // store final in the reserved location
    $GBF_N[emptySlot] \leftarrow final$;
  // over download limit
  else
    $final \leftarrow AB_N[p]$;
    for $i = 0$ **to** $k - 1$ **do**
      $j \leftarrow h_i(p)$;
      if $GBF_N[j] ==$ NULL **then**
        if emptySlot == -1 **then**
          $emptySlot \leftarrow j$;
        else
          $GBF_N[j] \xleftarrow{\$} (0,1)^\lambda$;
          $final \leftarrow final \oplus GBF_N[j]$;
      else
        $final \leftarrow final \oplus GBF_N[j]$;
    endfor
    $GBF_N[emptySlot] \leftarrow final$;
endfor
for $i = 0$ **to** $m - 1$ **do**
  if $GBF_N[i] == NULL$ **then**
    // store random strings
    $GBF_N[j] \xleftarrow{\$} (0,1)^\lambda$;
endfor

Algorithm 3: Download.Compute

Input: GBF_N, GBF_T, c, e, k, H
Output: R_N, R_T
1 Initialise $R_N, R_T \leftarrow \{0\}^\lambda$;
2 **for** $i \in [k]$ **do**
3 $p \leftarrow h_i(c)$;
4 $R_N \leftarrow R_N \oplus GBF_N[p]$;
5 $q \leftarrow h_i(e)$;
6 $R_T \leftarrow R_N \oplus GBF_T[q]$;
7 **endfor**
8 return R_N, R_T;

Algorithm 4: Download.Validate

Input: S_U, R_N, R_T, $\hat{c^*}$, $\hat{e^*}$
Output: FK
1 Initialise ${E_N}^*, {E_T}^* \leftarrow NULL$;
// element protection keys
2 $K_N[c^*] \leftarrow PRF_{S_U}(\hat{c^*})$;
3 $K_T[e^*] \leftarrow PRF_{S_U}(\hat{e^*})$;
// fail on inconsistent download states
4 ${E_N}^* \leftarrow Dec_{K_N[c^*]}(R_N)$;
5 ${E_T}^* \leftarrow Dec_{K_T[e^*]}(R_T)$;
6 $FK \leftarrow hash({E_N}^*||{E_T}^*)$;

data owner maps each $n \in N$ and $t \in T$ to the GBF's locations and adds shares of cipher key elements only to the locations hashed from $n \leq l_N$ and $t \leq l_T$ (line 4-17 in Algorithm 2). All other locations are filled with cipher payload messages (line 18-31 in Algorithm 2) or random strings (line 35 in Algorithm 2). Although the server holds the GBFs, the standard secret sharing technique ensures the shares of the cipher key elements and payload messages are of the same λ-length hence indistinguishable, and each share reveals no information about the secret. OblivSend never discloses the expiration metadata to a smart contract either, and the only information exposed is the system default download limit in M_N and M_T populated in Upload.Generate (line 5, 11 in Algorithm 1), which is visible to all OblivSend users hence not introducing extra security risks.

Further, OblivSend enforces the expiration control at the client side, hence neither the server nor the smart contract is aware of the decryption outcomes to deduce the download limits. If a server wants to learn the download limits, it needs to decrypt R_N and R_T (line 4, 5 in Algorithm 4): if the decryption succeeds, the download state c and e used are under the download limits; otherwise, over the limits. In order to decrypt R_N (or R_T), the server requires $K_N[c^*]$ (or $K_T[e^*]$) that is and output of PRF on S_U. A security

parameter σ is used to generate S_U, and the probability of a server to find $K_N[c^*]$ is $2^{-\sigma}$, which is negligible for all sufficiently large σ. The smart contract does not have R_N or R_T, thus not able to decrypt them and infer the download limits.

Server integrity auditing. OblivSend makes use of the public audit-ability of a blockchain to precautionarily detect if a server has tampered its download state. Recall the immutability assumption of a standard blockchain in § 3.4.3, the world download state store on a blockchain is immutable, which means that the hash state $(\hat{c^*}, \hat{e^*})$ fetched based on current download count and timestamp via the smart contract is authentic. If a server uses a forged download state, e.g. a replay attack, we must have $c \neq c^*$ and/or $e \neq e^*$. Note that the server computes R_N and R_T using its internal download state c and e, and the secret shares from mapped locations of c and e recover $C_{E_N}[c]$ (or $AB_N[c]$ if $c > l_N$) and $C_{E_T}[e]$ (or $AB_T[e]$ if $e > l_T$), which are encrypted using $K_N[c]$ and $K_T[e]$ respectively in Upload.Generate (line 7-8, 13-14 in Algorithm 1). The probability that a client can decrypt R_N using $K_N[c^*]$ is $Pr(c \neq c^* : K_N[c] = K_N[c^*])$. Since $K_N[i]$ is an output of a PRF on S_U and $\hat{c^*}$ (line 6 in Algorithm 3.5), the server's probability to trick the client into decrypting R_N computed from an illegitimate download number c is negligible if PRF is collision resistant.

Though OblivSend does not prevent a server from manipulating its state, it detects the inconsistency and prevents file decryption from happening. As long as one honest client reports the server's misbehaviour to the data owner after a decryption failure, the data owner can re-upload the file and share a new URL, hence such attack gains no information and little value.

File confidentiality. OblivSend uses hash values rather than the original download numbers and time units to protect the two key elements E_N and E_T, so that a semi-honest client cannot recover the key elements and decrypt a file by guessing a download state that has not expired without non-trivial computation. In Download.Validate, the client already has R_N, R_T, S_U, and wants to get E_N and E_T not using $(\hat{c^*}, \hat{e^*})$ but trying a download state (u, v) that is as minimal as possible. In order to recover E_N and E_T, the client needs $K_N[c]$ and $K_T[e]$, hence $\hat{c}$ and $\hat{e}$ (line 2-5 in Algorithm 4). Therefore, the client needs to find a pair of u and v so that $\hat{u} = \hat{c}$ and $\hat{v} = \hat{e}$ and this event occurs with at most negligible probability if the hash function is collision resistant.

General metadata protection. OblivSend sends a hash file URL, upload and download timestamps and two maps of system allowed download numbers and time units to a smart contract, and the smart contract also updates and keeps a world download state that includes a current count and elapse time unit corresponding to a file in a public ledger on a blockchain. However, OblivSend only publishes a digest of hashes to a ledger, therefore, no extra trace will be visible to adversaries because of the preimage resistance property of a hash function. Recall the anonymous network assumptions OblivSend makes in §3.4.3, users' IP addresses are garbled when communicating with a server or a smart contract, and OblivSend encrypts metadata such as file name, size, type in the same way as *Send* does [1]. OblivSend addresses general metadata privacy in file sharing.

3.6 Implementation and Evaluation

3.6.1 Implementation

We implement a command-line-based OblivSend prototype on Hyperledger Fabric programmed in Go. Since the purpose is not to measure the AES efficiency or network transmit throughput, we use a demo string instead of a real file in the prototype. We have the following settings: security parameters $\sigma = 128$, $l = 256$ (SH-256). The size of N (i.e. number of elements to be added to GBF_N) is 100 (maximum download number), and the size of T is 2016 (maximum 7 days with a minimum of 5 minutes), which follows *Send*'s setting. Security parameter $\lambda = 64$ that yields a negligible false positive probability 2^{-64}. The number of hash functions $k = 5$, and the length of GBF $m = \frac{kn}{ln2} \approx 1.44kn$ that is optimal [72], both of which can be passed dynamically during experiments. H is a family of 128-bit hash functions.

We present pseudo-codes of key functions performed by each party, and Pseudo-code 1 is an example that shows how a peer node executes a smart contract in response to a client's download request. Each peer in a blockchain network hosts a copy of the ledger, and OlibvSend can perform read and write operations against the ledger by invoking a smart contract which queries the most recent value of the ledger and returns it to OblivSend and/or updates the value of the ledger. In Pseudo-code 1, the smart contract takes a hash url and a download timestamp from the client's request, and queries the

most recent value of the ledger. It first validates its local copy of the ledger and then returns the value, meanwhile increments the download count and synced an updated digest to the world state.

```
func smartContractDownload
input: Request
output: Response, nImg, tImg
  hashedUrl, dl = read hashedUrl from Request
  state = get world state from the ledger(hashedUrl)
  if error
    return error "Failed to get world state", NULL
  countDigest = state.countDigest
  if countDigest == NULL
    return error "Download count not found in world state", NULL
  hashCount = Hash(count) //download count stored in local memory
  if hashCount != countDigest
    return error "Inconsistent download count", NULL
  ulDigest = state.ulDigest
  if ulDigest == NULL
    return error "Upload timestamp not found in world state", NULL
  hashUl = Hash(ul) //upload timestamp stored in local memory
  if hashUl != ulDigest
    return error "Inconsistent upload timestamp", NULL
  etUnit = F(dl - ul)
  hashNMap = Hash(nMap) //number image array stored in local memory
  if hashNMap != state.nMapDigest
    return error "Inconsistent number image array", NULL
  hashTMap = Hash(tMap) //time image array stored in local memory
  if hashTMap != state.tMapDigest
    return error "Inconsistent time image array", NULL
  nImg = nMap[count] //get number image by download count
  count += 1
  countDigest = Hash(count)
  tImg = tMap[etUnit] //get time image by elapsed time unit
  digest = new DigestStruct(hashedUrl, countDigest, ulDigest, nMap_Digest, tMap_Digest)
  Put digest to the world state
    if error
          return error, NULL
  return success, nImg, tImg
```

Pseudo-code 1: Function smartContractDownload

Pseudo-codes of other main functions are enclosed in Appendix. Note that the blockchain architecture is out of scope hence the implementation of Hyperledger Fabric network, e.g. how to build the network, and how to install smart contract (aka chaincode) onto peers, are not detailed.

3.6.2 Evaluation

The prototype Hyperledger Fabric network is deployed on a single cloud instance. It consists of one orderer node and two peer nodes, and runs in a docker-local container environment. We created an instance in a Research Cloud on Ubuntu 18.04 LTS (Bionic) amd64 (with Docker) version, with 2 virtual CPUs and 8 GB RAM. We used a Dell Precision 5530 with Intel i7 2.6GHz CPU, 32 GB RAM to make a SSH connection to the cloud instance.

Running Time. During the experiments, we measured the total running time of the upload phase and download phase respectively. The running time of the upload phase starts when a data owner initiates the upload request and ends after sending data to

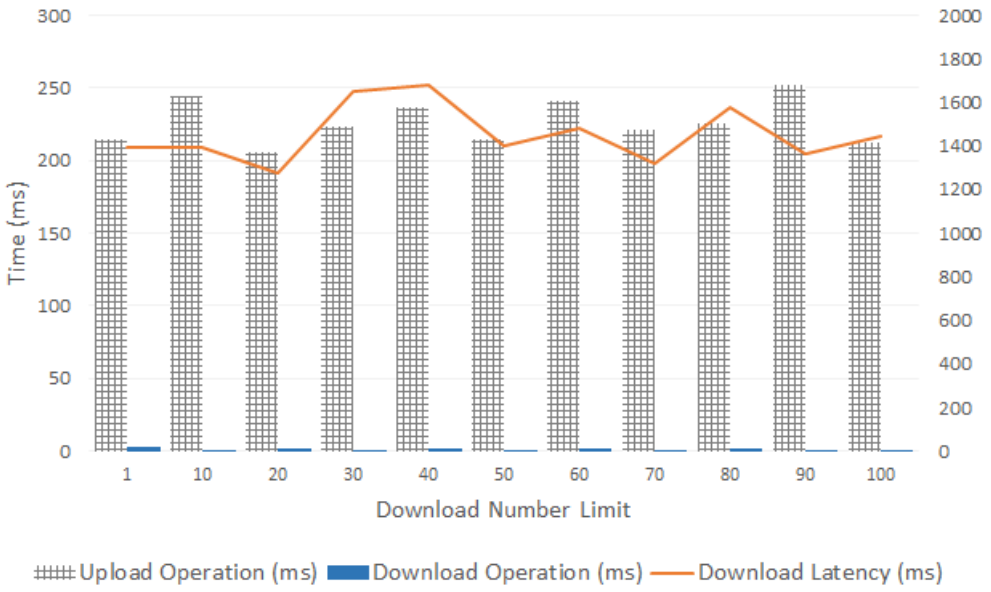
a smart contract, which includes the time consumed to build the GBFs, generate keys, maps, a hash URL, and all underlying data processing. We also measured the total download time that starts from a client extracting the file URL and ends immediately after the client outputting the decrypted demo, which includes operation time at both the server side (computing the GBFs) and the smart contract side (looking up for hashes and communication overhead). The communication overhead when querying and syncing the world state in a ledger on a blockchain is recorded separately as latency time. We do not, however, include communication overhead at the smart contract end during the upload phase as it occurs after data owner successfully uploads the data hence that does not explicitly impact user experience. We set one client to request a download for each experiment and ran 1000 independent experiments with random download limits, and the calculated average time is recorded in table 3.3.

Phase	Operation	Latency	Running Time
Upload	220.5283	NA	220.5283
Download	2.5385	1470.6911	1483.3837

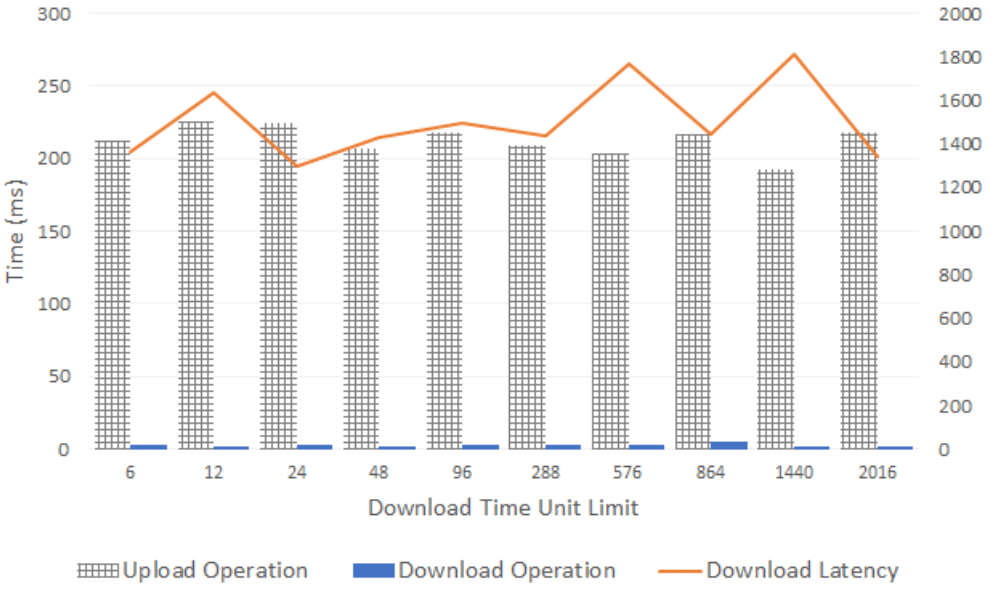
TABLE 3.3: Average Total Running Time (ms) (record size = 1000).

We test the download number limit by fixing the time limit as the maximum of 7 days, and experiment download time limit separately by setting the download number limit as the default number 100. As can be seen in Figure 3.4a and Figure 3.4b, the total running times are consistent over different download limit settings due to the undifferentiated construction of GBFs. The average total running time that Data Owner takes to finish all upload operations is 220.53 milliseconds, which is a satisfying performance. With regard to the download phase, the average total running time 1483.38 milliseconds, of which the latency time constitutes over 99 percent. However, our proposed protocol is an privacy-enhanced extension of secure large file sharing system, and in a real scenario, downloading a 25-MB file over mobile can take 8 seconds and a 1-GB file takes 3 minutes on fixed broadband just for transit (according to [80], world-wide average download speed for fixed broadband and mobile are 74.64 and 30.47 Mbps as of March 2020), not considering all underlying data processing operations. Therefore, an average of 1.5-second delay does not have substantial impact from the perspective of user interaction and experience.

Scalability. OblivSend is a secure and impermanent file sharing system that is dedicated for confidential and intimate files, hence we refer to existing secure file sharing



(a) Change Number Limit Only



(b) Change Time Limit Only

FIGURE 3.4: OblivSend Running Time

or messaging tools [49, 81] to determine pragmatic maximum allowed download number and time, which are the size n (or t) of the GBFs. We consider the maximum set size $n = 5000$ and the maximum set size $t = 8640$ (i.e. expiration time from 5 minutes to 30 days). According to [72], the running time of building a GBF increases almost linearly in the set size, and the estimated running time of building the GBFs on a 128-bit security parameter ≈ 1.3 seconds when uploading a file, which is not noticeable by an end user during the course of uploading a file.

Remark: The experiment demonstrates feasible application of GBF and smart contract in secure file sharing solutions, but a limitation of OblivSend is that it does not process download requests in a concurrent context. The order of a client raising a download request decides the current download count hence yields different decryption outcomes. An existing solution [78] is to have the server, not the client to publish data to a blockchain. However, this is not feasible in our protocol as OblivSend audits server's integrity using the immutable ledger on a blockchain as discussed in 3.5.1. The challenge related with concurrent access remains to be solved and future work for this topic.

3.7 Conclusion

This chapter proposes OblivSend, a lightweight privacy-preserving file sharing web service that for the first time protects expiration metadata from the server and meanwhile ensures E2EE by adopting cryptography protocols like GBF and novel Smart Contract technology. We have successfully implemented the design and built a prototype over a Hyperledger Fabric network. We also conducted experiments to evaluate its performance. The result is as expected and encouraging, demonstrating that OblivSend has stronger security without performance sacrifice. OblivSend precautionarily detects malicious mutations of a server's internal state, and we consider a simple scenario that one client requests download after another, both of which is open for further work.

Chapter 4

Privacy-Preserving File Sharing with Oblivious Expiration Control

In this chapter, we revisit privacy-preserving file sharing systems with oblivious expiration control and focus on further improving the privacy performance to make server(s) fully oblivious of file accessed and corresponding expiration control metadata. In OblivSend, although a server is oblivious of the expiration control, which is at the client's end; however, a curious server may deduce the expiration status of a file if it is no longer requested.

This chapter therefore proposes OblivShare, a privacy-preserving file sharing scheme to proactively protect the expiration control metadata, which is a subsequent privacy concern along with the trending feature in the industry to set download constraints of shared files as additional level of security control. The scheme is based on ORAM for secure computation that 1) supports file expiration at users' control, 2) hides expiration metadata from the server, 3) server is fully oblivious of file access pattern and expiration state of a file. We demonstrate that our protocol has a complexity poly-logarithmic to the number of files while achieving privacy of metadata.

4.1 Introduction

Users sharing files with other users over the Internet are common practices today. However, data leakages and mass surveillance projects [3, 5] have drawn public attention of

the vulnerability and sensitivity of personal data, and in turn promoted privacy awareness of users. Further, existing regulations and acts to protect personal data [43, 44], also impose on service providers to grant individuals control over their private information. Therefore, sharing files securely and privately is becoming a fundamental requirement.

In order to achieve secure file sharing, systems and services have been developed to support end-to-end encryption (E2EE) [52], using which, a user encrypts file content before it leaves their device and only authorised users are able to decrypt the file. However, E2EE does not appear to fully protect the privacy of user or file metadata, and a file can stay in servers indefinitely. Recent innovative services [1, 11] provide impermanence of data store on top of E2EE, which offers extra security control to users over the files they share: setting files to expire after a certain amount of time or number of downloads. On the one hand, such services incorporate two most desired features, **E2EE** and **ephemeral**, which meets personal needs of more secure connections and intimate sharing; on the other hand, limitations are also apparent: 1) Users send expiration control metadata (expiration metadata for short in the rest of this work), i.e. download number and time limits to check if a file has expired, to servers in plaintext, which can be used to deduce the popularity and sensitivity of specific file(s). 2) Expiration control is at servers' hand and users have to fully trust a service to honestly check if a file has expired.

Inadequate discussion has since occurred to understand the privacy of expiration metadata. Therefore, we aim to propose a new protocol to solve this emerging problem with practical values. This is a first attempt to focus on secure file expiration control, and the proposed protocol has not yet been implemented in real cloud environment. Security and performance analysis are provided in this chapter, and we consider real experimental evaluation to illustrate the performance in the future.

4.1.1 Motivation

"If you have enough metadata, you don't really need content", "we kill people based on metadata" [45, 46]. Sharing a file resembles calling or messaging someone from the perspective of metadata exposure, hence metadata privacy in file sharing is also concerning. While increasing service providers provide expiration control on top of E2EE, a gap exists in both industry and academia. To illustrate the motivation of

hiding expiration metadata and oblivious file sharing, we present some privacy issues even with E2EE file sharing systems.

Sensitivity derived from expiration metadata. Alice is an oncologist, and shares files with patients and other contacts in an E2EE system. Alice shares medical records with her patients and sets each to expire after 1 download, and other files without expiration conditions. With knowledge of the expiration metadata, a curious server learns that Alice shares some files with strict access, hence deduces they are sensitive. Bob is a patient of Alice and downloads his report from the system. With Alice's identity and the sensitivity of the file, the server thus infers Bob is suspected to have cancer without decrypting the report.

4.1.2 Summary of Contributions

We now propose OblivShare, a secure and ephemeral file-sharing system that for the first time provides users with advanced and oblivious expiration control. OblivShare puts forward a new framework of a file-sharing scheme that not only supports comprehensive file expiration control, but is also expiration-metadata-private and oblivious. This is a generic solution that can be integrated into file sharing services to address metadata privacy issues. To understand our contributions, we now outline the main challenges OblivShare aims to address.

Challenge 1: how to achieve expiration control over protected expiration metadata? We define expiration metadata as: 1) User-set download constraints, i.e., download number and time limits to facilitate expiration control. 2) Internal download state, i.e., current download count used to compute expiration control outcome. Users are not able to download a file if it has expired. To the best of our knowledge, whereas many scholars focus on protection of general security control metadata in file sharing such as user identity and access pattern [82], there is no prior research aiming to prevent leakage of expiration metadata, hence a gap exists to address such expiration-metadata-privacy.

Challenge 2: how to make download requests of a specific file indistinguishable from servers? Only hiding the expiration control process, outcome and metadata is not sufficient, as a server can still infer that a file has expired if the specific file has

Goal	Technique
Expiration metadata privacy	Secret sharing
Oblivious expiration control	Secure two party computation
Oblivious file sharing	ORAM
User IP addresses	Anonymous network, e.g., Tor

TABLE 4.1: Overview of techniques to achieve the goals.

not been accessed for a long time. A server not fully oblivious of the file sharing process learns which file is accessed for each download and can reasonably deduce the expiration metadata.

4.1.2.1 Contribution

OblivShare supports E2EE meanwhile protects the expiration metadata through the entire course with oblivious file access and expiration control. Our goals and techniques are summarised in Table 4.1.

With OblivShare, we propose an efficient secure file sharing scheme that not only achieves lightweight system design on top of ORAM (we present performance analysis in §4.5 that proves OblivShare has poly-logarithmic complexity), but also enables expiration control while hiding expiration metadata.

Overall, our contributions are:

1. We are the first to address metadata privacy issues in file sharing systems that support expiration control. User-defined download constraints are hidden from servers through the entire course of file upload, sharing and download. Internal download state is also protected by secret sharing between servers, therefore the servers cannot directly learn file expiration status.

2. We use synchronised tree-based ORAMs to store both file content and metadata, which hides file access patterns from servers, hence the servers cannot distinguish which file and how many times is requested so as to deduce file expiration status and further expiration metadata.

3. We are the first to use secure computation for oblivious expiration control, which not only guarantees that a single server cannot manipulate the expiration control result, but is also efficient to implement using garbled circuits.

4. We also demonstrate that our scheme has negligible extra computation and communication overhead on top of a primitive ORAM file sharing system, which requires one interaction with users hence not sacrifices user experience.

4.2 Related Work

4.2.1 Existing secure file sharing

Ephemeral content sharing is a highly pursued feature in industry [1, 11, 14, 49, 52, 83, 84]. With certain expiration control, users are confident that what they share is only accessible to dedicated users for limited time or number of times, and never stay in a server for longer than necessary and become a vulnerability later.

Table 4.2 compares several existing secure file sharing applications or web services. We organise the comparison by the following properties: 1) Does it support E2EE ? 2) Does it support file expiration? 3) Does it hide expiration metadata? 4) Is the server oblivious of file access and expiration control if applicable?

[1], [11] and [14] claim to offer zero-knowledge E2EE. [11] (premium) provides client-side encryption that keeps a public key protected encryption key in a key sever (isolated from the file storage server), and only a client's private key can decrypt the encryption key. [14] uses OpenPGP encryption and the file encryption key consists of a server secret (generated by the server) and a client secret (generated by the sender). Services such as [51] and [50], though do not support E2EE, but provide an addition layer of password security on top of server-side encryption. A user can double encrypt files or folders by setting a password, and share it to clients outside the service. [49] also offers E2EE for file attachments and has been developing its "Expiring Messages" feature.

Our solution is aiming to address the security weakness of existing systems mentioned in §4.1 with a good balance of desired features and cost.

4.2.2 ORAM for file storage

Oblivious RAM (ORAM) [85] is an attempt to hide a user's access pattern from service providers meanwhile supporting extra operations. Traditional ORAM schemes usually

Product	E2EE	Time Limit	Number Limit	Hide Expiration Metadata	Oblivious Server
OblivShare	✓	✓	✓	✓	✓
OblivSend [16]	✓	✓	✓	✓	✗
Firefox Send [1]	✓	✓	✓	✗	✗
DropSecure [11]	✓	✓	Future	✗	✗
SendSafely [14]	✓	✓	✗	✗	✗
WhatsApp [49]	✓	Future	✗	✗	✗
Digify [50]	✗	✓	✗	✗	✗
Dropbox [51]	✗	✗	✗	NA	✗

TABLE 4.2: Secure File Sharing Services.

have worst-case communication complexity linear to their capacity and block size even with amortized communication cost [86], and their single client setting [22, 85, 86] is not suitable for file sharing. Multi-user ORAM schemes are promising designs that can be applied in file sharing, but unfortunately, very few of such works exist. Among those that support file sharing, GORAM [87] is a system that guarantees anonymity of users and obliviousness of data access; but it does not protect the owner of a file. PIR-MCORAM [88] is a multi-user ORAM-based file sharing system, but has a very high overhead hence liner worst-case complexity. There are other ORAM schemes that focus on malicious users but do not readily support file sharing [89, 90].

At the best of our knowledge, none of the existing ORAM schemes, either hide access patterns and/or user identities or not, with linear or poly-logarithm complexity, has addressed expiration control.

4.3 Preliminaries

OblivShare makes black box use of secure two party computation, and also follows ORAM paradigm for metadata and file storage.

Notation. We define parameters, entities, denotations in OblivShare in Table 4.3.

4.3.1 Secure Computation

The Millionaires' Problem first described by Yao [25] enables to solve the following problem: Alice and Bob have their own secret inputs, which are their wealth x^A and x^B

Notatic	Description
λ	ORAM's statistical security parameters
ts_U	a timestamp that denotes the upload time
ts_D	a timestamp that denotes the download time
ts_{Exp}	a timestamp that denotes the expiration time, that is $ts_U + t$
D	an array of data content stored in ORAM
x	an positive integer that denotes a file index in ORAM, up to the ORAM file number bound, and $D[x]$ is the data stored in ORAM
Exp	an expiration policy that stores download constraints and state indexed by x
$[s]$	a secret share of s
N	the number of real data blocks in ORAM
h	the height of the ORAM tree, that is $\lceil log_2 N \rceil$
θ	a threshold of timestamp difference that is accepted by OblivShare

TABLE 4.3: OblivShare Notation

million, respectively. Yao's protocol enables that Alice and Bob can compute a function $f(x^A, x^B) \longrightarrow (y^A, y^B)$ such that Alice learns only its function output y^A while Bob knows only y^B, i.e., who is richer, and nothing else about the other party's wealth.

Since Yao's secure computation protocol was proposed, researchers have advanced a number of variations and extensions to address different scenarios. Recent secure multi-party computation (MPC) solutions includes private sorting [91], private computational geometry [92], private voting [93], and private data mining [94, 95] etc.

4.3.2 ORAM

OblivShare deploys ORAM for oblivious data storage and retrieval. More specifically, we use ORAM for secure computation [96–98], so as to ensure oblivious data access in MPC applications. There is a class of tree-based ORAM schemes [21, 22, 96] that are efficient for practical implementations especially in MPC, among which, we consider Circuit ORAM [96] as an appropriate scheme for our setting because of its competitive performance. Comparing to schemes like SqrtORAM [97] and Floram [98], Circuit ORAM client has complexity that is poly-logarithmic to the number of files, and also reduces the circuit size comparing to Path ORAM [21] and SCORAM [99]. Circuit ORAM is a tree-based ORAM. To store N files, Circuit ORAM constructs a binary tree with height $h = \lceil log_2 N \rceil$. The tree is composed of tree nodes, each of which has three blocks with fixed block size; apart from that, it also has a stash (up to the stash size

Algorithm 5: ORAMAccess

Input $op, idx, data$

Output $returnData$

1. $label \leftarrow PositionMap[idx]$
2. $\{idx||label||returnData\} \leftarrow ReadnRemove(idx, label)$
3. $PositionMap[idx] \leftarrow UniformRandom(0, ..., N-1)$ //update position map
4. **if** $op =$ "$read$" **then**
   (a) $data \leftarrow retrunData$
5. $stash.add(\{idx||PositionMap[idx]||data\})$
6. Evict()
7. Ouputs $returnData$

bound) that temporarily stores blocks that will be later evicted to the tree. Each block either stores the data of a file or is left empty. To store a file $D[x]$ in a file array D, a block contains the file index x, the file data $D[x]$, and its position that is the path from leaf to root. If a block is cached in the stash, the block stores the corresponding path that the block will be evicted onto. The file index x and its corresponding path p constitute a position map. We adapt Metal's protocol [48] as a underlying primitive for efficient and oblivious data access in S2PC.

Read from ORAM. To read a file, the two servers first check the file's leaf label (hence corresponding path) in the position map, then search for the block with the file index via a linear scan over both the stash and path. The servers then read the file block stored in the block. After reading the file, they randomly assign a new path to this block, put it back into the stash, and update the position map accordingly.

Write to ORAM. To write a file, the steps are similar until when the two servers add the block into the stash, and they replace it with the data to write provided by the user.

Stash eviction. Circuit ORAM performs a stash eviction for each read and write operation, at which stage, blocks cached in the stash are evicted to the ORAM tree to prevent stash overflowing. We do not elaborate the eviction algorithms of Circuit ORAM in detail here, but will illustrate rearrangement steps that are relevant to OblivShare. A generic Circuit ORAM data access operation is provided in Algorithm 5.

4.3.3 Synchronised inside-outside ORAM trees

We identify that the **synchronising inside-outside ORAM trees** technique used by METAL [48] is suitable for OblivShare. As has been introduced in 4.3.2, taking Circuit ORAM as an instance, each block in an ORAM tree contains the file index x, the file data $D[x]$, and its position. METAL, however, splits position map (i.e. index and path position) and actual file data, and stores them in two ORAM trees separately: one tree contains files' indices and positions stays inside S2PC procedures because it is small while the other tree that stores actual file contents stays outside S2PC. The two trees are maintained synchronised during initialisation and after each data access so that the file identifier and content can be found at the same position in the two trees. By doing so, the position of a file can be processed and revealed securely and efficiently in S2PC without loading large file data, and the block fetching and eviction of the actual file data are achieved by two protocols to keep the trees re-synchronised. METAL uses a secret-shared doubly oblivious transfer protocol to ensure that servers fetch the actual file data after revealing the position (in secret shares), and a distributed permutation protocol to track the movement of blocks after eviction and apply the rearrangement to according positions, without any servers learning the actual file's position.

In what follows, we provide some background knowledge of METAL's techniques relevant to our setting and describe more details in Appendix A.1.

Secret-shared doubly oblivious transfer. In order to get the actual file block outside S2PC, the two servers first process and reveal the file position inside S2PC, which means that the i-th block on the path p stores the position map and file data respectively in two ORAM trees. The S2PC then generates a list of keys for all the blocks on the path and outputs all these keys to Server 1 that stores the ORAM of actual file data, and Server 2 receives only one key corresponding to the actual file location i. Server 1 then needs to encrypt all the file blocks on the path p using the corresponding keys in order and re-randomise the encrypted blocks before sending them to Server 2. Server 2 uses its key received from the S2PC and decrypt blocks received from Server 1 to obtain the i-th block without either server getting the actual file location.

Distributed permutation. As has been mentioned in 4.3.2, stash eviction is called after every read or write after fetching a data block in ORAM. *Distributed permutation* [48, 97] captures the rearrangement of blocks, which is used when putting the read

block into stash before eviction and later evicting stash blocks to selected paths. The two servers in S2PC generate a permutation of an array of blocks, including the blocks in the stash, the block read and the block to write; then secret shares the permutation; and apply permutation shares accordingly. The result of the protocol is that the two ORAMs store the permuted blocks in the same location as per the updated file position map. By following this protocol, neither server learns the new position of the file after eviction, and neither server knows which permutation, read or write, is performed.

4.4 System Overview

In OblivShare, a data owner encrypts a file and sets the file to expire at certain conditions before uploading. OblivShare stores both the cipher file and expiration policy in a secure manner. When a client makes a download request to OblivShare, OblivShare first performs expiration control over download constraints and download state, then sends the cipher file to the client if the file has not expired. To understand how OblivShare fulfils these operations securely, we present an overview of OblivShare's design, threats and security goals.

4.4.1 System Architecture

A high-level framework of OblivShare is illustrated in Figure 4.1, which consists of two servers, a data owner and multiple clients:

- **Data Owner** sends upload requests to OblivShare, and shares the file index and file encryption key embedded in a URL to clients via secure channels.
- **client(s)** sends download requests to the servers, and receives results from OblivShare as per expiration check.
- **Servers** each takes its share of the requests as inputs to the S2PC, and together run S2PC procedures and send the outputs from S2PC to the clients. The servers also keep updated ExpCtrlORAM and DataORAM, which is explained in detail in §4.5.1.

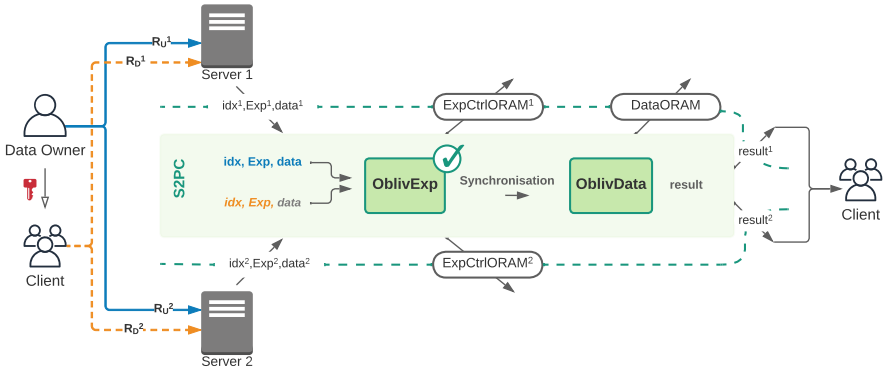


FIGURE 4.1: High level framework. A client sends its secret-shared request of file access to the two servers. The request is reconstructed and executed in S2PC.

OblivShare incorporates two major components: **OblivExp** for expiration Control and **OblivData** for file access. **OblivExp** is placed in front of **OblivData** to conduct expiration control. A client request first arrives at **OblivExp**, which checks whether the requested file has expired or not inside the S2PC by the two servers. If no, the request is sent to **OblivData** for a file access. If yes, the request is also dispatched to make the expiration control result indistinguishable to the servers, but in a manner to access dummy data instead. This is a loose description, and detailed construction is elaborated in §4.5.

OblivExp updates ExpCtrlORAM after each access and the changes to blocks as a result of stash eviction are applied to DataORAM during **OblivData** via synchronisation between ExpCtrlORAM and DataORAM.

4.4.2 Threat Model and Security Goals

4.4.2.1 Assumptions

OblivShare makes the following assumptions:

At least one server is honest. An attacker can compromise one of the servers in the two party secure computation while the other is not.

Key secrecy. A client does not reveal the URL with the key to adversaries.

Out-of-band communication. A data owner in OblivShare shares a URL with client(s) through third party secured channels of their own control, such as *Telegram* [52] and

Signal [13]. OblivShare only uses such out-of-band communication once at the sharing stage, which is a common practice of other secure file sharing systems [1, 51], keeping all other activities within OblivShare.

Anonymous Network. In order to hide other metadata during file sharing, OblivShare assumes the clients communicate with servers in an anonymous manner that does not reveal network information via existing tools such as Tor [12] or secure messaging [52] based on decentralised trust.

Secure communication. Each client establishes secure connections with each server, e.g., Transport Layer Security, so that data in transition are secured.

4.4.2.2 Threats

OblivShare considers the following threats:

1. A server can see the expiration metadata of a file. It enables the server to learn data sensitivity and popularity of the file, also deduces other valuable information of encrypted data, which has been explained in § 4.1.
2. A server on its own has control over its internal download state metadata, hence can forge the state, e.g. a small download count or an expiration timestamp that never expire.
3. A server can observe the file access pattern, hence the server is able to learn which specific file is accessed and the number of times the file has been accessed. If a file no longer receives download request, the server can deduce that the file has expired hence infer the user-set expiration metadata.
4. An attacker controlling a client tries to compromise the security of a file that has expired.
5. A client can forge its download timestamp so as to make an invalid download pass the expiration control check.

4.4.2.3 Security Goals

We now present security goals of OblivShare with respect to the threats given in §4.4.2.2.

1. **Expiration metadata privacy.** OblivShare ensures expiration metadata is totally at a data owner's control, and not visible in transit or at rest on either server.

2. **File confidentiality.** OblivShare ensures that neither server learns the actual file content; further, a compromised client cannot access the encrypted file and decrypt the content after the file has expired.

3. **Oblivious expiration control.** OblivShare ensures both servers are oblivious of the expiration control process. Though OblivShare does not prevent a server from manipulating its download state or a client forging its timestamp, the S2PC procedure for expiration control will fail if it detects compromised inputs to the S2PC. Hence such attack gains no information and little value.

4. **Oblivious file sharing.** OblivShare ensures neither server learns access patterns so that the servers are not able to infer if a specific file has expired hence expiration metadata.

5. **Download timestamp integrity.** OblivShare ensures that the download timestamp is independent of a client's input, but is controlled in S2PC.

6. **General metadata protection.** Recall the anonymous network assumptions OblivShare makes in §4.4.2.1, users' IP addresses are garbled when communicating with a server, and OblivShare addresses general metadata privacy in file sharing.

OblivShare guarantees the goals above based on common cryptographic assumptions. However, OblivShare does not address DOS attacks, which means OblivShare does not prevent a dishonest server from denying a valid download request even if the time has not expired or the number of downloads permissible has not been exceeded.

4.5 Detailed Construction

Note that in Circuit ORAM, the linear search of the file index happens within the S2PC, the real data is too large to process. We identify that METAL's synchronised ORAM trees [48] benefits our design to reduce the data accessed inside the S2PC. Below we introduce building blocks of OblivShare. In the following, we present our protocol assuming each file is an single file block for simplicity, but in practice, an uploaded file consists of multiple file blocks and is padded to have the same size.

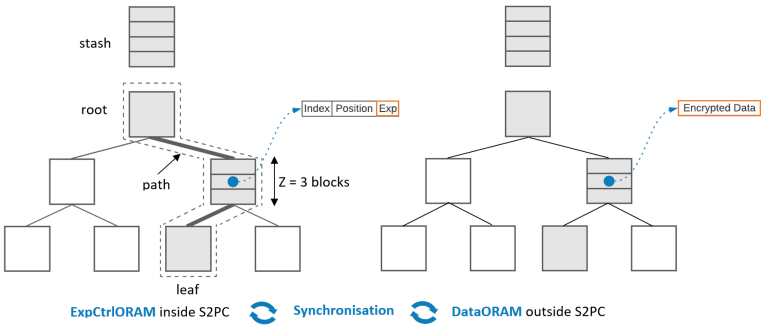


FIGURE 4.2: OblivShare has two tree ORAMs that store small metadata and large real data respectively and synchronised in ExpCtrlORAM and DataORAM.

4.5.1 Synchronised ORAM trees

OblivShare has two ORAM trees, an ExpCtrlORAM tree to store the metadata in a recursive manner and a DataORAM tree to store the actual file data. The two trees are synchronised so that the content and the metadata of a file are at the same location in DataORAM and ExpCtrlORAM.

ExpCtrlORAM is a set of trees that recursively stores small metadata, including: 1) the position map; 2) the expiration metadata, i.e. download constraints set by the data owner and the current download state. The ExpCtrlORAM is secret shared with two servers, and will be completed loaded and accessed inside the S2PC. OblivShare uses the standard recursive technique [48, 96] to store the metadata in ExpCtrlORAM, and for simplicity purpose, we refer to the last tree when using ExpCtrlORAM in the rest of this chapter.

DataORAM resembles ExpCtrlORAM's last tree but only stores the data (encrypted file). The DataORAM tree is stored on Server 1, and only relevant portion of the data structure will be loaded into the S2PC .

Figure 4.2 shows the ORAM structure in OblivShare and how two ORAMs are synchronised by storing corresponding metadata and data at the same location.

The read and write operations of Circuit ORAM still suffice data retrieval and update in ExpCtrlORAM but no longer fulfil fetching data from and putting data into DataORAM. OblivShare follows MTEAL's *secret-shared doubly oblivious transfer* (SS-DOT) [48] protocol to fetch the i-th block on path p from DataORAM. At the end of SS-DOT, Server 2 obtains the fetched block, i.e. the i-th block (encrypted under ElGamal) in the array, without either server learning i. We describe the details relevant to OblivShare

in Appendix A.1.1. OblivShare also follows *distributed permutation* to make sure ExpCtrlORAM and DataORAM are re-synchronised after each eviction by tracking and permuting block movements. At the end of *distributed permutation*, Server 1 stores the permuted file blocks in DataORAM in the same location as metadata blocks in ExpCtrlORAM. By following this protocol, neither server learns the new location of the evicted block. We give more details of how permutation is created, shared and applied in Appendix A.1.2.

4.5.2 OblivExp for Expiration Control

4.5.2.1 Upload a file

During the initialisation stage at the Data Owner's end, it generates an expiration policy Exp locally, which is a description of file download constraints and download state that is shared among the two servers. For example, a policy of a file that expires after 10 downloads and on "21 June 2021 21:21:21" has a policy "File Index x : 10, 21-06-2021T21:21:21, 0" where x is the file index, 10 is the download count (e.g. expire after 10 downloads) chosen by the Data Owner, t_{Exp} is the expiration timestamp derived based on upload timestamp (e.g. 18 June 2021 21:21:21) and download time setting (e.g. expire after 3 days) chosen by the Data Owner, and 0 is the initial download count. The Data Owner also encrypts a file using a secret key before the file is sent to Server 1. Algorithm 6 shows the secure computation during Upload.

Remark. After ExpCtrlORAM's stash eviction, evicted blocks are at new locations but the blocks in DataORAM are not rearranged. To synchronise DataORAM, we use *distributed permutation protocol* by applying the same rearrangement to data blocks in DataORAM. We will elaborate how OblivData ensures the data blocks in DataORAM is still synchronisation in §4.5.3.

4.5.2.2 Download a file

When a client requests a file download by a file index, the two servers search the file index in ExpCtrlORAM, then retrieve the path p and the expiration policy Exp of the file following primitive ORAM read process. The two servers then access DataORAM

Algorithm 6: OblivExp.Upload

Input of S1: $[x]^1, [Exp]^1, [ExpCtrlORAM]^1$
Input of S2: $[x]^2, [Exp]^2, [ExpCtrlORAM]^2$
Output: $[ExpCtrlORAM]^1, [ExpCtrlORAM]^2$

1. $x \leftarrow [x]^1 \oplus [x]^2, Exp \leftarrow [Exp]^1 \oplus [Exp]^2, ExpCtrlORAM \leftarrow [ExpCtrlORAM]^1 \oplus [ExpCtrlORAM]^2$; //reconstruct
2. $p \leftarrow PositionMap[x]$; //select a path
3. $(ExpCtrlORAM', \perp) \leftarrow ORAMAccess(ExpCtrlORAM, x, Exp)$, $ExpCtrlORAM \leftarrow ExpCtrlORAM'$; //the distribute permutation protocol will be invoked before and after ExpCtrlORAM's stash eviction
4. $[ExpCtrlORAM]^1 \leftarrow \$, [ExpCtrlORAM]^2 \leftarrow ExpCtrlORAM \oplus [ExpCtrlORAM]^1$; //secret share *ExpCtrlORAM*
5. Output to $[ExpCtrlORAM]^1$ to S1 and $[ExpCtrlORAM]^2$ to S2 respectively.

on the client's behalf and return the file block back to the client via secret shares if the expiration control check passes; otherwise a dummy instead. Note that the S2PC locates the i-th block on the path p in ExpCtrlORAM is the block for file x by a linear search, hence can access the encrypted data of file x in the i-th block of the same path p in DataORAM due to the synchrony between two ORAMs.

During the expiration control check, the two servers inside the S2PC determine a mutually agreed download timestamp (e.g. agree on a deviation threshold θ and then take a mean of the two timestamps from each server), and run the S2PC to compare download constraints to internal download state.

The two servers in the S2PC also update lock status of a file requested on the fly to indicate if the file is being accessed. The file is locked until the data, either the encrypted file or a dummy, has been successfully returned to the client.

To ensure the servers do not know if a file has expired, the S2PC appends a dummy block, and secret share the location i related to this dummy block instead of the actual block if a file has expired (step 11 in Algorithm 7).

Remark. Note that the two servers cannot simply fetch the i-th block in DataORAM, after revealing i in ExpCtrlORAM as the location i is related to the block history, i.e. a location i that is closer to the root level of the ORAM is more likely to have been accessed and evicted recently, and vice versa [100].

Algorithm 7: OblivExp.Download

Input of S1: $[x]^1, [ts_D]^1, [ExpCtrlORAM]^1$
Input of S2: $[x]^2, [ts_D]^2, [ExpCtrlORAM]^2$
Public Input: θ
Output: $[ExpCtrlORAM]^1, [ExpCtrlORAM]^2$, $[i]^1$, $[i]^2$

1. $ts_D \leftarrow agree(ts_D^1, ts_D^2, \theta)$; //agree on a download timestamp
2. **if** $ts_D = \perp$ **then stop.** //the procedure stops if the agreement fails
3. $x \leftarrow [x]^1 \oplus [x]^2, ExpCtrlORAM \leftarrow [ExpCtrlORAM]^1 \oplus [ExpCtrlORAM]^2$;
4. $locked \leftarrow Exp(status)$; //check the current lock status
5. **if** $locked = TRUE$ **then stop**.
6. **else** $locked \leftarrow TRUE$; //change the lock status to locked
7. $(ExpCtrlORAM', \{p, Exp\}) \leftarrow$
   $ORAMAccess(ExpCtrlORAM, x, \perp), ExpCtrlORAM \leftarrow ExpCtrlORAM'$;
   //run an ORAM read operation to get the expiration policy
8. $i = search(x, p)$; //determine the i-th location on path p that stores Exp
9. $r = isValid(Exp(count), Exp(number), Exp(ts_{Exp}), ts_D)$; //expiration check
10. **if** $r = TRUE$ **then** $Exp(count) += 1$; //increments the current download count
11. **else** $i \leftarrow |stash| + 3h + 1$; //add a dummy at the end of the array and point i to it
12. $locked \leftarrow FALSE$ and $Exp(status) \leftarrow locked$; //reset the lock status
13. Generate $[ExpCtrlORAM]^1$,
    $[ExpCtrlORAM]^2 \leftarrow ExpCtrlORAM \oplus [ExpCtrlORAM]^1$
14. Generate $[i]^1$, $[i]^2 \leftarrow i \oplus [i]^1$
15. Output $[ExpCtrlORAM]^1$, $[i]^1$ to S1 and $[ExpCtrlORAM]^2$, $[i]^2$ to S2 respectively.

To make Upload and Download indistinguishable, expiration policy is constructed as {File Index: download number, expiration timestamp, download count, download timestamp, lock status}, hence {File Index: download number, expiration timestamp, 0, $\perp$, FALSE} for Upload and {File Index: $\perp$, $\perp$, $\perp$, download timestamp, $\perp$} for Download.

4.5.3 OblivData for File Access

In what follows, we show how the two servers in combination fetch and put a file, which is the same for each ORAM Upload and Download.

4.5.3.1 Fetch data from DataORAM

In §4.5.2, we already provide a solution of indistinguishable file fetch regardless of expiration status by appending a dummy block. After OblivExp completes the expiration control, either passed or failed, it proceeds the request to OblivData that fetches a block, either a real data block or a dummy depending on the expiration control result, in the form of different values of the location i. As has been briefed in §4.5.1, at the end of *SS-DOT*, Server 2 received ElGamal [101] cipher-texts of the data block at the i-th location on path p in DataORAM, with neither server aware of i. Upon ElGamal decipher, the result is either the actual file content or the dummy encrypted under a file encryption key (shared by a data owner to dedicated clients during the share stage), which is finally returned to the client who can further decrypt the result. Algorithm 15 in Appendix A.1.1 shows how the *SS-DOT* protocol works in OblivData.

Remark. Note that in Algorithm 15, j is independent of i as a result of shuffle, hence Server 2 is not aware of i all through the course.

4.5.3.2 Evict Data to DataORAM

After ExpCtrlORAM's stash eviction, positions of blocks are updated in ExpCtrlORAM, hence OblivData needs to ensure the corresponding real data blocks in DataORAM are rearranged in the same manner. To guarantee that the two ORAMs are still synchronised, OblivShare tracks the block movements in ExpCtrlORAM and apply the same changes to DataORAM. We use *Distributed Permutation* again during this stage following a similar manner of METAL to re-synchronising trees after each eviction.

Algorithm 16 in Appendix A.1.2 demonstrates how the re-synchronisation is achieved by tracking the movement of blocks in ExpCtrlORAM and applying the same permutation to DataORAM, hence Server 1 stores the blocks in the corresponding locations.

4.5.4 Security Guarantees

We now present security guarantees of OblivShare with respect to the goals given in § 4.4.2.3.

1. **Expiration metadata privacy.** OblivShare hides expiration metadata from both servers yet is able to enforce the expiration control by using secret sharing. The standard secret sharing technique ensures the shares of the expiration metadata are of the same length hence indistinguishable, and each share reveals no information about the secret (line 1 in Algorithm 6). Also, OblivShare uses ElGamal encryption for data blocks stored in ORAM, which prevents the leakage of sensitive expiration metadata from the servers during the entire course.

2. **File confidentiality.** Data owner of OblivShare encrypts a file before uploading the file to servers and only shares the private key to authorised clients. In addition, all data blocks in ORAM are ElGamal encrypted hence the servers cannot decrypt the actual file content without non-trivial computation. OblivShare also prevents invalid access to expired file by returning a dummy instead of the encrypted file (ensured by line 11 in Algorithm 7) so that a compromised client cannot retrieve a file that has expired even with the private key.

3. **Oblivious expiration control.** During OblivExp and OblivData, OblivShare uses S2PC protocols to perform expiration control (line 7-11 in Algorithm 7 and the first stage of SS-DOT that samples random keys in line 2(a)-2(d) in Algorithm 15). The security of S2PC guarantees neither server learns or tampers the expiration control result.

4. **Oblivious file sharing.** The obliviousness of ORAM guarantees that file access patterns are hidden from the servers.

5. **Download timestamp integrity.** (line 1-2 in Algorithm 7) The security of S2PC guarantees neither server learns the input timestamp of the other hence cannot modify the actual timestamp or fabricate a new timestamp that is used in the following expiration control operation (line 9 in Algorithm 7).

6. **General metadata protection.** OblivShare makes the anonymous network assumptions in §4.4.2 that users' identities and their online activity are encrypted during client-server communications through existing secure tools [12, 52]. OblivShare also encrypts metadata such as file name, size, type in the same way as it does for a file hence addresses general metadata privacy in file sharing.

Non-guarantees. As stated in §4.4.2, OblivShare does not address DOS attacks by a server, neither protects from malicious server(s), which means OblivShare does not guarantee the availability of a file if a dishonest server denies a valid download request.

4.5.5 Performance

We consider the system supports N files in total (for simplicity, each file is a single block hence N data blocks) with block size S in DataORAM. As a result of METAL's synchronised inside-outside ORAM trees, the cost for accessing small metadata blocks in ExpORAM is negligible and considered as constant comparing to accessing large data blocks in DataORAM [48]. We use $\mathcal{O}_\lambda(\cdot)$ to present the complexity, while N_{block} is polynomially bounded by λ. We parameterise to have $\frac{1}{N^{\omega(1)}}$ failure probability that is the same as Circuit ORAM [96].

The amortised computational cost of Circuit ORAM is $\mathcal{O}_\lambda(S + \log_N^2) \log N) \cdot \omega(1)$, and OblivShare has minimal additional cost on top of Circuit ORAM. The file access, i.e., read and write operations in OblivShare's `Upload` and `Download` are indistinguishable and have the same cost that includes the cost of Circuit ORAM, SS-DOT and distributed permutation. During `Download`, OblivShare's expiration control incurs additional cost.

The cost for creating download timestamp (line 1-2 in Algorithm 7) is $\mathcal{O}_\lambda(1)$. Expiration control is independent of data blocks in DataORAM and has constant cost $\mathcal{O}_\lambda(1)$ (line 3-12 in Algorithm 7). The total cost of SS-DOT, including Server 1 fetching blocks (line 1(a) in Algorithm 15), the S2PC generating keys (line 2(d) in Algorithm 15), Server 1 encrypting blocks (line 3(b) in Algorithm 15), and the maximum cost of Server 2 decrypting blocks (line 4(b) in Algorithm 15), is linear to the number of blocks fetched on the path and in the stash (with constant size) hence is $\mathcal{O}_\lambda(\log N)$. Distributed permutation also has total cost linear to the blocks on the paths and in the stash (line 1 in Algorithm 16), therefore $\mathcal{O}_\lambda(\log N)$.

Table 4.4 summaries the above cost and the total computational complexity of OblivShare for both `Upload` and `Download` is $\mathcal{O}_\lambda((S+\log_N^2) \log N) \cdot \omega(1)$. `Upload` and `Download` have the same computational complexity and communication complexity following Metal's protocol [48], which is linear to the file size S and poly-logarithmic to the number of files N.

Stage	Computational Complexity
`Upload`	$\mathcal{O}_\lambda((S+\log_N^2)\log N+\log N)\cdot\omega(1)=\mathcal{O}_\lambda((S+\log_N^2)\log N)\cdot\omega(1)$
`Download`	$\mathcal{O}_\lambda((S+\log_N^2)\log N+\log N+1)\cdot\omega(1)=\mathcal{O}_\lambda((S+\log_N^2)\log N)\cdot\omega(1)$

TABLE 4.4: Total computational complexity for `Upload` and `Download` stages.

4.6 Conclusion

We proposed OblivShare, a lightweight privacy-preserving file sharing scheme that for the first time protects expiration metadata together with file access patterns from servers meanwhile ensures oblivious expiration control by adopting cryptography protocols including Secure Computation and ORAM, which fill the gap in the previous chapter. We prove that our protocol can achieve its security goals without additional cost that the computation and communication complexity is poly-logarithmic to the number of files.

Chapter 5

Secure and Privacy-Preserving Multi-Party Path Planning Framework

The previous experiment of practical privacy-preserving file sharing focused on enhancing metadata privacy atop of E2EE, underscoring secure data management and access control in digital communication. With the increasing deployment and growing popularisation of unmanned aerial vehicles (UAVs), such as surveillance, package delivery, and environmental monitoring from both civilian and even military use scenarios, protecting their sensitive operational data and metadata is essential to address emerging privacy and security challenges. The secure computation primitives established in the earlier chapter provide a solid cryptographic foundation to extend privacy-preserving techniques to collaborative UAV operations, aiming to contribute to the growing field of research addressing privacy concerns in drone applications and provide a foundation for the development of secure and collaborative UAV systems across diverse domains with a balance between robust security measures and operational feasibility.

This chapter presents a novel approach that employs SMPC techniques atop generic path planning algorithms to address these security and privacy challenges. SecuPath, a secure framework that leverages cryptographic protocols for secure communication and computation, enables multiple entities to jointly compute optimal drone paths without revealing their private inputs to any party. By integrating this privacy-preserving layer

onto generic drone path planning algorithms, we ensure the optimality of the planning process while significantly elevating the privacy and security standards of UAV path planning operations. This framework not only preserves the confidentiality and privacy of sensitive data, but also fosters collaboration in scenarios where data sharing is essential, such as in urban airspace management or disaster response. Moreover, this chapter analyse the communication and computation overhead introduced by the SMPC and demonstrate the protocol remains practical.

5.1 Introduction

5.1.1 Background

Unmanned Aerial Vehicles (UAVs), commonly known as drones, have witnessed widespread adoption in both civilian and military domains due to their versatility and capability to perform a wide range of tasks. In civilian applications, drones are utilised for tasks ranging from surveillance and package delivery to environmental monitoring, providing efficient and cost-effective solutions [102, 103]. Simultaneously, in military operations, UAVs play a pivotal role in reconnaissance, surveillance, and tactical operations, enhancing situational awareness and operational capabilities [104, 105].

However, the increasing deployment of drones into various applications has brought forward a number of security and privacy concerns, particularly in the realm of data exchanged during path planning. Path planning is a critical aspect of drone operations, influencing their trajectory and optimal navigation. However, existing path planning algorithms often require data communication between multiple parties, such as data owners and edge servers, introduction potential threat to the privacy of sensitive information shared during path planning. Issues such as GPS spoofing [106] and interception of communication channels [107] have raised alarms regarding the confidentiality of data exchanged between drones and their control systems.

In response to these concerns, several legal frameworks and regulations have been established to safeguard the privacy of data collected and used in drone operations, whether it be location data, images, or other identifiable details. Regulations such as the General Data Protection Regulation (GDPR) [43] and specific guidelines provided by aviation

authorities [108] set strict standards for the collection, storage, and processing of data, emphasising the need for responsible and secure handling of sensitive information to ensure that only the necessary and lawful information is shared among stakeholders to protect against unauthorised access or data breaches.

5.1.2 Motivation

This chapter addresses the pressing need for enhancing the security and privacy of drone path planning, focusing on the multi-party setting involving data owners, edge servers, and other relevant entities. The objective is to enable secure collaboration between multiple parties involved in the drone ecosystem without compromising the confidentiality of their respective data.

The motivation behind this research question stems from the need to address emerging and sophisticated security and privacy threats. As UAVs become integral to diverse applications, ensuring the confidentiality, integrity, and availability of data exchanged during path planning is crucial. We present some typical concerns to demonstrate why data privacy is imperative for collaborative UAV path planning.

Prevent Path Inference Attacks. In the context of collaborative UAV search and rescue operations, multiple organisations collaborate to plan paths for UAVs searching in a disaster-stricken area. Adversaries may exploit shared path planning data to analyse the trajectories of UAVs involved in search and rescue missions, which could reveal sensitive information, such as the targeted locations or critical waypoints, potentially compromising the safety and security of the mission. Preventing path inference attacks can ensure that trajectories contributed by collaborating entities remain confidential, maintaining the effectiveness and security of the collaborative UAV efforts in disaster response scenarios.

Protect against Exposure of Operational Constraints. Consider a scenario where multiple organisations collaborate on a UAV mission that involves surveillance in a sensitive facility. The UAVs have specific operational constraints, including altitude limits, speed restrictions, and predefined flight paths within the secure facility to prevent unauthorised access. An adversary may attempt to exploit the collaborative path planning

data to extract specific operational constraints, e.g., altitude limits, to identify vulnerabilities in the surveillance system. Safeguarding the operational constraints is vital to prevent potential adversaries from gaining insights into the security measures in place, protecting the privacy of UAV operational parameters, and adhere to regulations or policies specific to the facility.

Preserve Identity Privacy in Collaborative Path Planning. Imagine a consortium of engineering companies collaborates to deploy UAVs for inspecting critical infrastructure, each contributing expertise and data for inspecting specific sections of infrastructure. The shared path planning data may contain details about the expertise of each company, their inspection methodologies, and specific sections of infrastructure under their responsibility. An adversary could exploit the path planning data to deduce the involvement of specific engineering company in the consortium, which could lead to potential competitive disadvantages, industrial espionage or risks related to intellectual property. Preserving data privacy is crucial to prevent potential identification of specific collaborators involved in the consortium.

5.1.3 Challenges

There are multiple perspectives and respective approaches to achieve privacy-preserving UAV path planning, and this research will focus on solving data privacy issues by addressing the following main technical challenges:

Challenge: how to preserve data privacy in collaborative UAV path planning while maintaining effective communication and collaboration between drone operators and edge servers? There are several security concerns that we aim to address during path planning: 1) Data confidentiality and privacy: The data collected by drones such as terrain information, ecological parameters and weather conditions that are used in optimising the UAV's route, the operational constraints such as maximum flight height, restricted areas must be kept private and secure to protect the privacy of customers. 2) Data integrity: The data sent and received by drones, such as current geographic coordinates, planned way-points in the optimal path, must be accurate and verified to ensure correct trajectories and mission objectives. 3) Authentication and access control: It is crucial to verify the identity of the party that requests access to the edge server, and hide collaborator identity data, such as responsibilities and access

permissions of individuals or organisations involved in the planning process. 4) Secure communication: All communications between the parties must be encrypted. To the best of our knowledge, whereas many scholars have developed frameworks to optimise UAV path planning [109], there is no prior research focusing on preserving data privacy in the entire path planning life cycle, hence a gap exists to address such path planning data privacy.

In summary, this research aims to establish an effective privacy-preserving path planning system to ensure anonymous and secure communication among collaborators, and maintain the confidentiality and integrity of shared data. It will not design novel cryptographic primitives, nor access control mechanisms.

5.1.4 Contributions to knowledge

We now propose SecuPath, a innovative approach to address privacy challenges in collaborative UAV path planning using multi-party secure computation. The key contributions can be outlined as follows:

1. **Privacy-Preserving Collaborative Framework** We introduce a novel collaborative framework that integrates SMPC techniques into UAV path planning. This framework allows multiple entities to collaboratively plan UAV paths while preserving the privacy of sensitive input data, such as the drone's current location and planned path, from exposing to the each other or other unauthorised parties.

2. **Enhanced Security Mechanisms** We incorporate cryptographic techniques to ensure the integrity and authenticity of the transmitted data, effectively protecting against prevalent risks in wireless communication environments, such as data tampering and GPS spoofing.

3. **Balanced Efficiency and Security** We provide a detailed analysis of the communication and computation complexities introduced by the SMPC and cryptographic operations, demonstrating the protocol's practical feasibility. It enhances security and privacy performance while maintaining operational efficiency, making it suitable for real-world UAV applications with stringent performance and resource constraints.

4. **Application to Diverse UAV Scenarios** The design of the privacy-preserving approach allows for applicability and adaptability of the proposed protocol across versatile UAV scenarios, such as environmental monitoring, search and rescue, and agricultural surveillance. Its security mechanisms can be adjusted based on the specific requirements and threat models of different applications, offering a versatile framework for secure UAV path planning.

5. **Scalable Multi-Party Secure Computation Protocol** We also approve that our proposed privacy-preserving framework is a scalable multi-party secure computation protocol tailored to the requirements of collaborative UAV path planning. This ensures that the privacy-preserving mechanisms are practical for real-world applications.

5.2 Literature Review

This literature review delves into existing research on privacy-preserving path planning and secure computation, offering insights into prior work that informs and contextualises the current study.

5.2.1 Related Work

5.2.1.1 Privacy-preserved location-based services

Privacy concerns in the domain of UAVs path planning have spurred considerable research efforts, with a focus on safeguarding sensitive data during planning processes. In this context, [38, 110–112] employ differential privacy mechanisms [28] to protect sensitive trajectory data. Their work addresses the challenge of concealing location information, providing a foundation for privacy preservation in UAV trajectory planning.

Building on privacy-centric approaches, [113] extended the exploration to cooperative route planning. Their framework delves into preserving privacy, particularly regarding intermediate waypoints, through cryptographic techniques. This work discusses privacy in collaborative settings.

Recent contribution [114, 115] proposed frameworks based on homomorphic encryption to ensure the confidentiality of private data and compute over encrypted data efficiently. Their work demonstrates the viability of leveraging advanced cryptographic techniques to facilitate UAV collaboration without compromising individual confidentiality.

5.2.1.2 Secure UAV communications and computations

In parallel, cryptographic techniques play a pivotal role in enhancing drone security by ensuring secure communication channels, verifying data integrity, and protecting sensitive identity. [116] provides a comprehensive overview of Public Key Infrastructure (PKI)-based security mechanisms tailored for UAV systems using a unique public-private key pair assigned to each drone and central authority. [117] explores the efficacy of Hash-based Message Authentication Code (HMAC) authentication protocols in securing UAV communication networks to verify the integrity and authenticity of messages exchanged between UAVs and ground control stations (GCS). Moreover, [118] discusses the integration of Advanced Encryption Standard (AES) encryption to protect sensitive data stored on drones and transmitted over communication channels secure communication between the UAV and GCS, addressing implementation challenges and optimisation strategies for efficient encryption in resource-constrained UAV environments.

[119, 120] contributed secure communication and computing protocols tailored explicitly for UAVs' computational offloading strategy. The protocols empower offloading large computing tasks to multiple entities that compute tasks in the presence of an eavesdropper while maintaining secure communications. [121] proposed path planning algorithms to determine and verify positions in a secure manner while producing short path lengths in a reasonable processing time. [122] formulated secure trajectory planning to guarantee that either the robot moves from a source to destination along the nominal trajectory or that attacks remain detectable.

Additionally, Identity-Based Encryption (IBE) simplifies key management by using entities' identities as public keys, offering a promising approach for secure communication, access control and authentication in drone systems. [123] proposes a security model based on Identity-Based (IB) authentication scheme for UAV-integrated HetNets.

Notation	Description
D	Drone
ES	Edge Server
λ	statistical security parameters in S2PC
V	the set of vertices (locations) in a graph presentation
E	the set of edges (paths) in a graph presentation
G	the graph representation of the geographic area, where $G = (V, E)$ with weights representing distances or costs
U	the currently considered vertex in the graph
$L_{current}$	current location of the drone
$L_{destination}$	destination of the drone operation
sk_{HMAC}	shared secret key for HMAC
$\|V\|$	the number of vertices in the graph
$\|E\|$	the number of edges in the graph
I	the number of iterations required by Dijkstra's Algorithm

TABLE 5.1: SecuPath Notation

To the best of our knowledge, there is no existing framework that directly apply secure computation to UAV path planning. This chapter endeavors to contribute to this research field by proposing a practical, scalable, and efficient mechanism employing multi-party secure computation atop generic algorithms for UAV path planning.

With SecuPath, we propose a privacy-preserving path planning framework that not only allows anonymous and secure path planning in collaborative settings, but also achieves lightweight system design on top of Dijkstra's Algorithm.

5.3 Preliminaries

SecuPath makes black box use of secure multi-party computation, and adapt from the classic Dijkstra's algorithm to ensure optimality in path planning.

Notation. We define parameters, entities, denotations in SecuPath in Table 5.1.

5.3.0.1 Multi-Party Secure Computation

Existing multi-party secure computation protocols provide a foundation for privacy preservation during collaborative path planning, which include cryptographic techniques to ensure secure collaboration among multiple entities.

The Millionaires' Problem first described by Yao [25] enables to solve the following problem: Alice and Bob have their own secret inputs, which are their wealth x^A and x^B million, respectively. SMPC enables that Alice and Bob can compute a function $f(x^A, x^B) \longrightarrow (y^A, y^B)$ such that Alice learns only its function output y^A while Bob knows only y^B, i.e., who is richer, and nothing else about the other party's wealth.

Since Yao's secure computation protocol was proposed, researchers have advanced a number of variations and extensions to address different scenarios. Recent secure multi-party computation (MPC) solutions includes private sorting [91], private computational geometry [92], private voting [93], and private data mining [94, 95] etc.

5.3.0.2 Dijkstra's Algorithm

We consider generic path planning algorithms for UAVs that can efficiently deploy secure computation techniques. Among several well-established algorithms [124–126], we choose Dijkstra's Algorithm, a fundamental path-finding algorithm renowned for its efficiency in determining shortest paths within weighted graphs. Leveraging a priority queue for exploration, it guarantees optimality while systematically updating the costs to reach each node [127].

The algorithm starts by initialising the distances from the source (which is the starting node) to all other nodes in a graph to infinity and the distance from the source to itself to 0. The weighted directed graph is represented as an adjacency list. Next, a priority queue is created, where each node is inserted with its current distance as the key. Then the algorithm enters a loop that continues until the priority queue is empty. In each iteration, the node with the minimum distance, which is extracted from the priority queue, is selected for further exploration. For each neighbor of the selected node, the algorithm explores potential paths by checking whether the distance from the source to the neighbor node can be improved by going through the selected node. If a shorter path is found, the distance is updated. This process continues until all nodes have been processed, and the algorithm ensures that the final distances represent the shortest paths from the source node to all other nodes in the graph. Given the source node, destination node, and the predecessors obtained from the Algorithm 8, the Algorithm 9 outputs the list of nodes representing the shortest path from the source to a destination node.

Algorithm 8: Main Algorithm

Input $graph, source$
Output $distances, predecessors$

1. $distances, predecessors \leftarrow initialise(graph, source)$; //initialise the distance
2. $Q \leftarrow priorityQueue(graph.nodes, distances)$; //create a priority queue and insert all nodes with their distances
3. **while** Q **is not** $empty$ **then**

    $u \leftarrow extractMin(Q)$; //extract the node with the minimum distance from the priority queue

    for $v, edgeWeight$ **in** $neighbors(u, graph)$

        $tentativeDistance \leftarrow distances[u] + edgeWeight$; //calculate the tentative distance from the source to the neighbor through the current node

        if $tentativeDistance < distances[v]$

            $distances[v] \leftarrow tentativeDistance$; //update the distance of the neighbor

            $predecessors[v] \leftarrow u$; //set the current node as the predecessor of the neighbor

            $updatePriorityQueue(Q, v, tentativeDistance)$; //update the priority queue with the new distance of the neighbor
4. **return** $distances, predecessors$;

Algorithm 9: Reconstruct Shortest Path

Input $source, destination, predecessors$
Output $path$

1. $path \leftarrow []$; //initialise an empty list
2. $currentNode \leftarrow destination$; //from the destination node
3. **while** $currentNode$ **is not None then** //until the source node

    $path.insert(0, currentNode)$; //inert nodes at the beginning of the list

    $current \leftarrow predecessors[currentNode]$; //backtrack by following the predecessor links
4. **return** $path$;

It's important to note that the efficiency of Dijkstra's Algorithm is greatly enhanced when using a priority queue data structure to efficiently extract the node with the minimum distance in each iteration. The "relax" function is a key part of the algorithm where the distances are updated if a shorter path is discovered.

As a conclusion of literature review, the privacy issue in UAV path planning has been explored in the last several decades; however, none of the existing provides a holistic

approach to privacy-preservation throughout the entire path planning life cycle. To fill the gap, this chapter will try to apply the cryptographic primitives to construct a privacy-preserving secure and anonymous path planning scheme.

5.4 System Overview

In this section, we propose a privacy-preserving path planning system by leveraging Dijkstra's Algorithm as the foundation for calculating the shortest path. A high-level framework will be presented, and how to satisfy the privacy-preserving requirements will be further discussed.

In SecuPath, the collaborative path planning process involves two main parties: a representative drone responsible for collecting location data, and an edge server tasked with performing path planning computations securely. While, for simplicity, we illustrate the system with one drone and one edge server, it's important to note that this configuration can be seamlessly expanded to accommodate multiple entities. The overall goal remains consistent, which is to ensure the privacy of sensitive location information while determining the optimal route for the drone or any additional entities involved.

To understand how SecuPath fulfils the path planning operations securely, we present an overview of SecuPath's design, threats and security goals.

5.4.1 System Architecture

A high-level framework of SecuPath is illustrated in Figure 5.1, which consists of one edge server, and one drone client (in this chapter, drone is used interchangeably with drone client):

5.4.1.1 Parties Involved

- **Drone Client** represent the UAV responsible for collecting environmental data from its surroundings and initiating path planning requests, acting as one of the parties in the S2PC protocol. It takes as input the initial and destination locations, along with the graph representing the environment, and any operational

constraints. The drone encrypts its input data using a secure encryption scheme to maintain the confidentiality of sensitive location information and then sends the encrypted request to the edge server. It receives and executes the path plan result from the edge server, and also updates the edge server periodically with its real-time status and any changes in the environment that might require a path re-calculation. The drone is equipped with light computing capabilities to perform preliminary data processing.

- **Edge Server** serves as the second party in the S2PC protocol, and execute the path planning algorithm securely to generate an optimal path. It takes the encrypted data received from the drone as the input and any additional information available to the server. The edge server performs the S2PC with the encrypted inputs, utilising cryptographic techniques to compute the shortest path without gaining knowledge of any sensitive information such as the actual locations. The output is an encrypted representation of the optimal path and the edge server sends the encrypted path back to the drone. It also monitors ongoing operations and suggest path adjustments in response to changing conditions or threats detected. The edge server acts as computational hubs with enhanced processing capabilities compared to drones, and execute resource-intensive tasks such as S2PC and path optimisation algorithms.

5.4.1.2 Data Flow

1. **Input Collection and Encryption** Drone Client collects initial environment data such as terrain information, obstacle positions, and weather conditions. The Drone processes its initial data locally to extract relevant features and reduce dimensionality, and encrypts using SMPC encryption techniques.

2. **Request Path Planning** Drone Client sends encrypted data securely to the Edge Server using cryptographic protocols requesting for an optimal path. Communication channels are safeguarded to prevent unauthorised access or tampering.

3. **Secure Path Planning** Edge Server receives the encrypted data from Drone and executes S2PC protocols to collaboratively compute optimal paths while preserving data privacy. Dijkstra's Algorithm is executed on the encrypted data to determine the most efficient paths. Edge server encrypts the output of the path planning

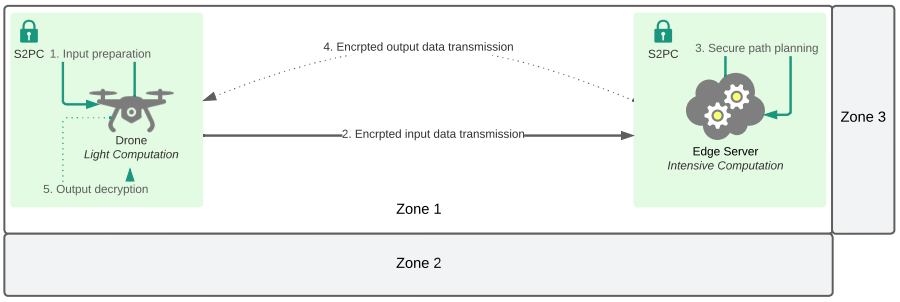


FIGURE 5.1: High level framework. A drone client sends its request of path planning with encrypted inputs to the edge server. The edge server executes the request in S2PC and sends back the encrypted path planning response. The drone then decrypts the encrypted path for further navigation.

computation using S2PC techniques. S2PC ensures that sensitive data remains encrypted throughout the computation process, preventing exposure to any single entity.

4. **Path Planning Response** Edge Server sends the encrypted path planning result back to the requesting drone securely.

5. **Result Decryption** The drone decrypts received paths locally using SMPC decryption techniques, ensuring data confidentiality is maintained until the final execution stage.

6. **Continuous Updating** During the entire course of the flight, the drone sends periodic updates to the edge server and the edge server can re-compute the path based on the updates and send updated commands to the drone, ensuring continuous secure communication.

As has been mentioned before, the system can be extended to include multiple drones. Also GCS can be added to the framework between a drone and an edge server that can perform intermediate computation and forward encrypted data to the edge server and send optimal path or route plan back to drones. Furthermore, as drones can fly over long distance covering a extensive area, the edge server is whichever responsible one in respective zones.

5.4.2 Threat Model

The following section provides a structured overview of the threat model for the proposed protocol by clarifying assumptions, identifying vulnerabilities, and guiding the implementation of security mechanisms, which establishes the security landscape for the secure computation protocol for drone path planning.

5.4.2.1 Assumptions

SecuPath makes the following assumptions:

Honest-but-Curious Entities SecuPath assumes that both the drone and the edge server are honest-but-curious, which means that they will follow the protocol but may attempt to learn sensitive information from the protocol execution.

Secure Communication Channel SecuPath assumes a secure communication channel between the drone and the edge server via existing encryption and authentication mechanisms such as Transport Layer Security [128] to ensure data in transit are secured.

Secure Environment Both the drone and the edge server operate in a secure environment against malicious physical or network access, e.g. firewalls, intrusion or hazard detection and response systems are implemented to prevent unauthorised access.

Network Resilience The communication network is assumed to have advanced defense mechanisms against DoS attacks, not only including rate limiting and traffic analysis but also redundancy to ensure the path planning services remain available despite potential network-based threats.

5.4.2.2 Threats

SecuPath considers the following threats:

1. An adversary may attempt to intercept the communication between the drone and the edge server and deduce sensitive information such as location data.
2. An adversary may tamper with the data in transit to disrupt the protocol execution, leading to incorrect optimal path calculation or false results.

3. An adversary can spoof the edge server's identity to send malicious commands to the drone or replay old computation results to disrupt the operation.

4. An adversary may exploit unintended sensitive information leakage through analysing timing patterns or power consumption.

5.4.2.3 Security Goals

We now present security goals of SecuPath with respect to the above threats.

1. **Confidentiality** SecuPath ensures sensitive information including path planning inputs, outputs and intermediate computations results remain confidential and not accessible to unauthorised parties throughout the protocol execution via secure computation.

2. **Integrity** SecuPath ensures the accuracy and completeness of information utilising cryptographic mechanisms to prevent data tampering during the communication between the drone and the edge server.

3. **Authenticity** SecuPath verified the identities of participating parties using secure channels and authentication mechanisms to prevent illegitimate parties from accessing the computation environment.

SecuPath guarantees the goals above based on common cryptographic assumptions. However, OblivShare does not address physical or DoS attacks, which has been clarified in the §5.4.2.

5.5 Detailed Construction

The proposed protocol SecuPath involves collaborative computation between the drone and the edge server to calculate the optimal path while preserving data privacy. The proposed protocol leverages existing SMPC technique for the implementation of Dijkstra's Algorithm, aiming to achieve the security goals while ensuring operational efficiency. Note that in SecuPath, the edge server is responsible for performing heavy computational tasks, such as circuit generation for secure computations and executing the

encrypted form of Dijkstra's Algorithm. The drone, on the other hand, participates in the S2PC protocol by sharing encrypted inputs and decrypting the received path planning results. This setup minimises the computational load on the drone, conserving its resources while leveraging the edge server's more substantial computational capabilities.

5.5.1 Path Planning in S2PC

SecuPath consists of several steps, which are depicted in the remaining of this section.

5.5.1.1 Input Preparation

The drone and the edge server each prepare their respective inputs for the Dijkstra's Algorithm computation and establish a secure communication channel.

The drone collects location data and initiates the path planning request by securely sending its current location (i.e., starting node) and destination, both encrypted, to the edge server.

The edge server prepares the encrypted graph representation for the shortest path computation by encrypting each edge weight using S2PC protocols, ensuring all data used in the computation is in encrypted form suitable for S2PC without revealing the actual weights or the structure of the graph to any party.

Note that in SecuPath, the drone establishes a secure session by initiating a secure session to the edge server and incorporate mutual authentication and nonce-based mechanisms to ensure message freshness as shown in Algorithm 10 and 11.

5.5.1.2 Secure Path Computation

1. *Initialisation.* The edge server initialises the distances from the starting point to all other vertices in the graph to infinity, except for the starting vertex, which is set to zero following Dijkstra's Algorithm, and all these distances are also encrypted.

2. *Circuit Generation.* Next, the edge server sets up the secure computational environment by generating the necessary S2PC circuits, e.g., a comparison circuit to compare two encrypted values to determine which is smaller, and an addition

Algorithm 10: Drone: Input Preparation

Input $L_{current}, L_{destination}$

Output $L^*_{current}, L^*_{destination}, nonce_{req}, value_{HMAC}$

1. $nonce_{req} \leftarrow generateNonce()$; //Generate a unique nonce for the session to prevent replay attacks
2. $L^*_{current} \leftarrow encrypt(L_{current}, sk_D), L^*_{destination} \leftarrow encrypt(L_{destination}, sk_D)$; //encrypt current location and destination using SMPC-compatible encryption
3. $value_{HMAC} \leftarrow generate(L^*_{current} + L^*_{destination} + nonce_{req}, sk_{HMAC})$ //generate HMAC for the encrypted data including the nonce to ensure integrity and authentication
4. **return** $L^*_{current}, L^*_{destination}, nonce_{req}, value_{HMAC}$;

Algorithm 11: Edge Server: Input Preparation

Input $L^*_{current}, L^*_{destination}, nonce_{req}, value_{HMAC}$

Output G^*

1. **if NOT** $verify_{nonce}(nonce_{req})$

   raise Exception; //verify the nonce to prevent replay attacks
2. **if NOT** $verify_{HMAC}(L^*_{current} + L^*_{destination} + nonce_{req}, value_{HMAC}, sk_{HMAC})$:

   raise Exception; //verify HMAC to ensure the integrity and authenticity of the received data
3. $G^* \leftarrow \{\}$; //proceed with preparing the encrypted graph for path planning
4. **for each** V // iterate over each vertex in the graph

   $E^* \leftarrow \{\}$;

   for $neighbor, weight$ **in** $edgesofV$ // iterate over each neighbor and edge weight connected to the current vertex

   $weight^* \leftarrow encrpyt_{SMPC}(weight)$; // encrypt the edge weight using SMPC

   $E^*[neighbor] \leftarrow weight^*$ // store the encrypted edge weight for the current vertex

   $G^*[V] \leftarrow E^*$; // store the dictionary of encrypted edge weights for the current vertex in the encrypted graph
5. **return** G^*

circuit to securely add two encrypted values to update distances. The circuits incorporate operations for graph traversal, distance calculation, and shortest path determination, hence are crucial for executing Dijkstra's Algorithm securely to find the minimum distance on encrypted data.

3. *Dijkstra's Algorithm Execution.* The edge server then executes Dijkstra's Algorithm using the encrypted graph and distances, employing the previously generated S2PC circuits for all comparisons and additions. The computation iteratively updates the encrypted distances to each vertex until the shortest path is determined without decrypting any intermediate results, thus preserving privacy. After computing the shortest distances, the edge server utilises secure circuits to reconstruct the path from the start vertex to the destination, ensuring that the path itself remains encrypted and private.

4. *Result Reconstruction.* Once the shortest path is determined, the edge server utilises secure circuits to reconstruct the path from the start vertex to the destination, and securely sends the encrypted path result back to the drone.

Algorithm 12 shows the secure computation on the edge server side.The edge server manage all the intensive computational tasks, including preparing the graph, generating S2PC circuits for secure operations, executing Dijkstra's Algorithm in an encrypted domain, and securely reconstructing the optimal path, while ensuring the process adheres to S2PC principles. This ensures that heavy lifting computations are offloaded to the edge server, which has more resources.

5.5.1.3 Result Retrieval

The drone receives the encrypted path and decrypts it to retrieve the optimal path planning result and act upon the optimal path for further navigation. This step is computationally light, aligning with the intention to minimise the drone's processing load.

The decryption process is a collaborative effort where the final computation result, i.e., the encrypted path can only be decrypted by the drone, which relies on the unique capabilities of SMPC to securely aggregate computations and reveal only the final output to authorised parties.

To secure the transmission of the computed path from the edge server to the drone, SecuPath uses HMAC for message integrity and authentication, along with a new nonce to prevent replay attacks.

Algorithm 12: Edge Server: Secure Path Computation

Input $L^*_{current}, L^*_{destination}, G^*$
Output $path^*$

1. $circuit_{comp} \leftarrow GenerateComparisonCircuit_{SMPC}()$; // generate SMPC circuits for secure computations
2. $circuit_{add} \leftarrow GenerateAdditionCircuit_{SMPC}()$;
3. $distances^*, predecessors^* \leftarrow initialise_{SMPC}(G^*)$; // setup initial encrypted distances and previous nodes using SMPC encryption technique
4. $Q \leftarrow G^*.keys()$; // create a priority queue to select the node with the lowest tentative distance
5. **while** Q **is not** *empty* **then**

   $U \leftarrow extractMin_{SMPC}(Q, distances^*)$; // extract the unvisited node with the smallest distance

   $Q.remove(U)$;

   for $neighbor, weight$ **in** $G^*[U]$ // explore the neighbors of the current node

   $tentativeDistance \leftarrow add_{SMPC}(circuit_{add}, distances^*[U], weight)$; // calculate tentative distance to neighbor

   if $compare_{SMPC}(tentativeDistance, distances^*[neighbor])$ // compare if the tentative distance is less than the current distance

   $distances^*[neighbor] \leftarrow tentativeDistance$; // update the distance and the predecessor node

   $predecessors[neighbor] \leftarrow U$;
6. $path^* = reconstruct_{SMPC}(predecessors^*, L^*_{current}, L^*_{destination})$; // securely reconstruct the path from the destination to the start
7. **return** $path^*$

Algorithm 13: Edge Server: Result Retrieval

Input $path^*$
Output $path^*, nonce_{resp}, path_{HMAC}$

1. $nonce_{resp} \leftarrow generateNonce()$; //Generate a new nonce for the session
2. $path_{HMAC} \leftarrow generate(path^* + nonce_{resp}, sk_{HMAC})$ //generate HMAC for the encrypted path
3. **return** $path^*, nonce_{resp}, path_{HMAC}$;

5.5.2 Security Guarantee

The proposed protocol ensures the confidentiality, integrity, and authenticity of path planning computations between the drone and the edge server, under the security assumptions outlined in § 5.4.2.

Algorithm 14: Drone: Result Retrieval

Input $path^*, nonce_{resp}, path_{HMAC}$
Output $path$

1. **if NOT** $verify_{nonce}(nonce_{resp})$

   raise Exception; //verify the nonce to prevent replay attacks

2. **if NOT** $verify_{HMAC}(path^* + nonce_{resp}, path_{HMAC}, sk_{HMAC})$:

   raise Exception; //verify HMAC to ensure the integrity and authenticity of the received data

3. $path \leftarrow decrypt_{SMPC}(path^*)$; //decrypts the path using its part of the SMPC decryption protocol to obtain the actual navigational instructions

4. **return** $path$

The proposed S2PC protocol for Dijkstra's Algorithm securely computes the shortest path in a semi-honest model, where the participating parties follow the protocol but are curious to learn additional information.

The detailed construction achieves the security goals mentioned in the Threat Model section as follows:

1. **Confidentiality** SecuPath utilise encrypted sessions and S2PC to ensure sensitive information remains confidential. Given the secure computation environment, path planning upon the encrypted data does not reveal any information about the underlying inputs values or intermediate computation result. The protocol ensures that neither the drone's current location and destination nor the details of the computed path are revealed to the edge server.

2. **Integrity** SecuPath uses nonce to prevent replay attacks, and HMACs for subsequent data exchanged between the drone and the edge server to ensure that the data hasn't been tampered with.

3. **Authenticity** SecuPath uses HMACs as a mechanism for authenticating the origin of the messages for each data communication. Derived from a shared secret key, only the legitimate parties that possess the shared key can generate or verify the correct HMAC for a message, therefore achieves mutual authentication during the secure communication session.

5.5.3 Performance Evaluation

We will provide an analysis of both communication and computation complexity for SecuPath. Given that SMPC introduces overhead due to the encryption, decryption, and secure computation processes, we will evaluate how these aspects impact the overall performance on top of the Dijkstra's Algorithm. We use the Big O notation $\mathcal{O}_\lambda(\cdot)$ to present the complexity, while

5.5.3.1 Communication Complexity

The primary communication occurs during **Input preparation** and **Result Retrieval**, when the drone sends its encrypted data to the edge server and receives the encrypted path from the edge server. Each involves a single round of communication, with the data size depending on the encryption scheme but typically proportional to the size of the input data. The complexity of secure session establishment is $\mathcal{O}(1)$, and cryptographic operations for encryption and HMAC generation/verification in Algorithm 10 and 11 also have a complexity of $\mathcal{O}(1)$. The complexity of transmitting the path planning requests and results can vary based on the path length but generally remains small. Therefore, the overall communication complexity is $\mathcal{O}(1)$ assuming fixed-size encrypted data (both the inputs and the output path result) and the SMPC encryption does not significantly alter the size of the transmitted data.

5.5.3.2 Computation Complexity

In **Secure Path Computation**, assuming the use of a priority queue for efficiency and given the number of iterations I is equal to the number of vertices V, i.e. $I = V$ iterations, the computation complexity of executing Dijkstra's Algorithm is $\mathcal{O}(|E| + |V| \log |V|)$. In the SMPC setting, the complexity remains the same, but the operations are performed on encrypted data, leading to increased computational overhead. Assuming the SMPC encryption operations have a complexity of $\mathcal{O}(\lambda)$, where λ presents the security parameter affecting SMPC encryption operations, thus, the effective computation complexity is $\mathcal{O}_\lambda(|E| + |V| \log |V|)$. During *result reconstruction*, the complexity is linear in the size of the path, $\mathcal{O}(p)$, where p is the path length, which is much smaller than E or V, hence minimal compared to distance computation. The complexity of

initial encryption and final decryption in SMPC in **Input Preparation** and **Result Retrieval** typically have a complexity of $\mathcal{O}(n)$, when n is the size of the encrypted inputs and outputs. However, this additional complexity is not substantial compared to the complexity of executing Dijkstra's Algorithm under SMPC, especially since n is typically much smaller than the size of the graph data ($|V|$ and $|E|$). Therefore, the overall computation complexity is $\mathcal{O}_\lambda(|E| + |V| \log |V|)$.

5.6 Conclusion

In this chapter, we presented a novel framework SecuPath for privacy-preserving collaborative UAV path planning leveraging SMPC and Dijkstra's Algorithm. Our framework addresses the critical need for protecting sensitive information such as the drone's current location, destination, and computed paths remains encrypted throughout the path planning operations, particularly in scenarios involving multiple parties with privacy concerns.

SecuPath addresses significant security concerns including privacy-preserving computation, integrity and authenticity, replay attack mitigation, and a balance between security and efficiency. By using a combination of SMPC and cryptographic techniques, despite the added computational overhead of secure computation, SecuPath offers robust security solution to secure drone path planning while maintaining reasonable complexity. SecuPath also ensures that the security measures do not excessively burden the drone's limited computational resources by offloading the computation intensive tasks to the edge server.

SecuPath also offers feasibility and effectiveness for real-world applications, with the capability to accommodate large-scale drone networks and can be easily adapted to involve multiple entities (e.g. multiple drones, edge servers, and GCS) while maintaining data privacy.

Chapter 6

Conclusion

This thesis has explored innovative approaches to enhancing privacy and security in various contexts of data sharing and exchange. By focusing on privacy-preserving file sharing with oblivious file expiration control and secure collaborative UAV path planning, the research contributes significant advancements in cryptographic and privacy-preserving systems.

The first contribution of this thesis is improving privacy performance atop of an existing E2EE file sharing web service *Send* in the industry, and proposes OblivSend that supports oblivious expiration control by concealing the expiration control metadata in GBF. It also demonstrates feasible application of Smart Contract in secure file sharing solutions to take advantage of the immutability of a blockchain to detect malicious mutations. OblivSend supports E2EE and allows users to set download constraints, meanwhile efficiently protects metadata privacy by making the server oblivious of the expiration control, which is entirely at the user's control.

Building on the concept of metadata privacy, the second contribution of this thesis is furthering OblivSend to design OblivShare, a privacy-preserving file sharing scheme that employs ORAM for secure computation. OblivShare addresses the need for privacy-aware users to ensure data security throughout the entire life cycle, preventing indefinite cloud storage. The scheme supports user-controlled file expiration, conceals expiration metadata from the server, and ensures the server is oblivious to file access patterns and expiration states. The protocol's complexity is poly-logarithmic to the number of files, demonstrating its efficiency while maintaining privacy.

Further, the third contribution of this thesis extends the scope of privacy-preserving techniques to collaborative UAV path planning that involves data sharing and exchange between client and server, addressing the increasing deployment of autonomous UAVs in sensitive applications. SecuPath leverages SMPC to enable multiple entities to jointly compute optimal drone paths without revealing their private inputs. This approach ensures data confidentiality and privacy, fostering collaboration in scenarios such as urban airspace management and disaster response. The framework's integration of cryptographic protocols with generic path planning algorithms preserves the confidentiality of sensitive data while maintaining the optimality of the planning process.

The research presented in this thesis has several important implications:

- **Enhanced Privacy and Security:** The proposed systems provide robust solutions to ensure the privacy and security of sensitive data exchanged in file sharing and UAV operations, addressing critical challenges in modern digital communication and autonomous systems.
- **User Empowerment:** By allowing users to control expiration constraints and protect metadata, the research empowers users with greater control over their data, fostering trust and confidence in digital services.
- **Scalability and Efficiency:** The proposed schemes demonstrate efficient performance and scalability, making them practical for real-world deployment in various applications.

This thesis has made contributions to the field of privacy-preserving systems by addressing key challenges in secure data sharing and UAV path planning. The proposed solutions demonstrate the feasibility and effectiveness of advanced cryptographic techniques in enhancing privacy and security. The suggested future research directions provide a comprehensive roadmap for advancing the field, ensuring that privacy and security remain at the forefront of digital innovation.

Through rigorous exploration and innovative solutions, this research has laid a solid foundation for future advancements in privacy-preserving data exchange systems, paving the way for more secure and trustworthy digital interactions in an increasingly interconnected world.

Chapter 7

Future Directions

Building on the foundational work in privacy-preserving data exchange in file sharing and UAV path planning, this chapter outlines some future directions that aim to advance the field of privacy-preserving technologies leveraging the underlying cryptographic primitives.

- **Scalability and Performance Optimisation:** Future research should focus on developing scalable cryptographic protocols and algorithms that can efficiently handle large datasets and accommodate increasing numbers of concurrent users. Techniques such as parallel processing, distributed computing, and optimised cryptographic primitives should be explored to minimise computational overhead and latency. For instance, A* algorithm's heuristic approach to guide the search towards the most promising paths first can effectively reducing the number of nodes evaluated and therefore lead to faster path calculations [129, 130], especially in scenarios with large datasets or complex network topologies where the path-finding task involves evaluating numerous potential routes. In addition, PySyft's optimisation techniques, such as parallelisation and resource allocation can enhance computational efficiency and minimise latency in SMPC operations to improve performance.

- **Verifiable Data Exchange Framework:** This thesis focuses on data and metadata confidentiality with semi-honest server setting. One future direction is to extend current research to develop protocols that are robust against active adversaries capable of arbitrary deviations from the protocol, ensuring that the system

remains secure even when some parties behave maliciously. This involves exploring advanced cryptographic primitives and constructions for verifiable computation, secure function evaluation, and authenticated data structures, also investigating the incorporation Byzantine fault tolerance [131] into privacy-preserving systems to ensure system reliability and security where components may fail and give conflicting information to different parts of the system. Such verifiable framework will ensure users can trust the data and results provided by the system, even in the presence of potentially malicious actors.

- **Secure Collaborative Framework and Protocols:** Advance research on secure protocols for multi-party computation and collaborative data analysis. Develop techniques that enable secure and private collaboration while multiple parties are involved with conflicting interests or varying levels of trust, ensuring fairness and integrity in joint computations. Expand the application of SMPC to enable privacy-preserving computations across distributed and heterogeneous environments. Investigate scalable and efficient protocols for secure data aggregation and collaborative decision-making. This includes exploring FL [132], where multiple parties collaboratively train machine learning models without sharing raw data, thereby preserving privacy and confidentiality.

- **Integrating Machine Learning with Privacy Preservation:** Explore the integration of privacy-preserving data exchange techniques with machine learning models.

  There are some potential topics under the above context:

  - Privacy-Preserving Machine Learning. This includes developing comprehensive frameworks to train machine learning algorithms on encrypted data, ensuring that the training process does not reveal sensitive information and compromise data privacy.
  - Secure Model Sharing and Verification. Investigate protocols for securely sharing trained models and verifying the correctness of models and their predictions. This involves developing robust algorithms that can withstand attempts to manipulate training data or model parameters to produce incorrect outputs, hence defend against adversarial manipulations on machine learning models in privacy-preserving settings.

– Federated Learning. As has been mentioned above, FL allows multiple clients to collaboratively train machine learning models without revealing personal data, which incorporates privacy preservation with distributed training and aggregation across a large population. Future research could focus on enhancing the privacy and security of FL by incorporating cryptographic techniques such as SMPC and differential privacy [28]. This would ensure that the contributions of each party remain confidential while achieving accurate and robust models.

Appendix A

An Appendix

A.1 METAL's synchronised inside-outside ORAM trees

A.1.1 Secret-shared doubly oblivious transfer

Let N be an array of the blocks in the stash and the $3h$ blocks on path p:

1. The two servers inside S2PC, generate n keys $k_1, \ldots, k_n$ such that S1 receives as output all these keys, and S2 receives only k_i. $n = |stash| + 3h + 1$.
2. For each $j \in 1, \ldots, n$, S1 uses k_j to encrypt 0 and m_j to obtain cipher-texts z_j and c_j respectively. S1 shuffles all the (z_j, c_j) pairs and sends them to S2.
3. S2 uses k_i to decrypt the first cipher-text of each pair: only one z_j , will decrypt to 0. It then decrypts the corresponding c_j and hence obtain m_i.

A.1.2 Distributed permutation

Recall that Circuit ORAM selects two paths during eviction, hence we need to track the movement of blocks in the stash and on the two paths, which has $|stash| = 6h - 3$ blocks.

Before each eviction, OblivShare appends a number tracker from 1 to $|stash| = 6h - 3$ to each block on the stash and two paths in ExpCtrlORAM inside S2PC. After the

Algorithm 15: OblivData.Fetch

Input of S1: $[i]^1, p, DataORAM$
Input of S2: $[i]^2$
Output: m_i

1. S1:
   (a) $blocks \leftarrow Fetch(DataORAM, p)$. //fetch all blocks in stash and on path p
2. S2PC:
   (a) $i \leftarrow [i]^1 \oplus [i]^2$
   (b) $blocks.Append(\perp)$; //add a dummy block at the end of the array
   (c) $n \leftarrow |blocks| + 1$; //so that $n = |stash| + 3h + 1$
   (d) **for** $j = 1$ **to** n **do** $k_j \xleftarrow{\$} (0,1)^l$; //generate n keys
   (e) Outputs $k_1, \ldots, k_n$ to S1 and k_i to S2.
3. S1:
   (a) $M \leftarrow \{\}[n]$ //initialise an array to store the encrypted pairs
   (b) **for** $j = 1$ **to** n **do** $(z_j, c_j) \leftarrow Enc_{k_j}(0, m_j))$; $M.add((z_j, c_j))$;
   (c) $M.Shuffle()$;
   (d) Sends M to S2.
4. S2:
   (a) $found \leftarrow FALSE$;
   (b) $p \leftarrow 1$ **while** $p \leq n$ and $!found$ **do**
      i. $(z_p, c_p) \leftarrow M[p-1]$; $z_p^{'} \leftarrow Dec_{k_i}(z_p)$;
      ii. **if** $z_p^{'} = 0$ **then** $found \leftarrow TRUE$; $m_p \leftarrow Dec_{k_i}(c_p) = m_i$; //$m_i$ is the i-th block on path p in DataORAM
      iii. $p++$;
   (c) Outputs m_i.

ExpCtrlORAM's stash eviction, the protocol extracts the trackers and construct an array of the numbers. Note that some numbers no longer exist as the attaching blocks are removed during the eviction. In order to generate a permeation of the same $|stash| = 6h - 3$ elements, OblivShare searches for the missing trackers using a linear scan and fill in the empty slots with unused numbers.

Below, we present how the two servers in secure computation put a block into the DataORAM's stash before eviction:

1. The S2PC places the following in an array: the blocks in the stash, the block read, and the block to write, which has $(|stash| + 2)$ blocks.

2. The S2PC finds that the k-th block of the stash is vacant, then generates a permutation σ^{read} or σ^{write}, which exchanges the k-th block with the read block for σ^{read} or the block to write for σ^{write}. As a result, the correct block is inserted into the stash (i.e. the first $|stash|$ blocks of the permuted array).

3. The S2PC secret shares the permutation (σ^{read} or σ^{write}) into two permutations σ^1 and σ^2, when $\sigma^2 = \sigma \circ (\sigma^1)^{-1}$. $\circ$ denotes composition of permutation and $\sigma \circ (\sigma)^{-1}$ is the identity permutation.

4. S1 re-randomise the blocks, apply the permutations σ^1, and sends the permuted blocks to S2.

5. S2 re-randomise the blocks received, apply the permutations σ^2, and sends the permuted blocks back to S1.

6. S1 stores the permuted blocks in the corresponding location in DataORAM.

Algorithm 16: OblivData.Sync

Input of S2PC: M

Output: $M^{'}$

1. **for** $i = 0$ **to** $|stash| + 6h - 4$ **do** $M[i].Append(i + 1)$; //attach a tracker to each block before ExpCtrlORAM's stash eviction
2. $M^{'} \leftarrow ExpCtrlORAM.Evict()$; //extract trackers after stash eviction
3. $trackers \leftarrow \{\}$; //initialise an array to store the missing trackers
   //do a linear scan to find numbers in $\{1, 2, \ldots, |stash| + 6h - 3\}$ that are missing
4. **for** $t = 1$ **to** $|stash| + 6h - 3$ **do**
   (a) $found \leftarrow FALSE$;
   (b) $k \leftarrow 0$ **while** $k \leq 18$ and $!found$ **do**
      i. **if** $M^{'}[k] = t$ **then** $found = TRUE$;
      ii. **else** $k++$;
   (c) **if** $!found$ **then**
      i. $trackers.add(i)$;
   (d) $t++$

   //do a linear scan to fill missing trackers into the empty slots
5. $r \leftarrow 0$
6. **for** $j = 0$ **to** $|stash| + 6h - 4$ **do**
   (a) **if** $M^{'}[j] = \perp$ **then** $M^{'}[j] = trackers[r]$; $r++$; //locate the empty slots and fill in missing trackers
   (b) $j++$
7. $\sigma \leftarrow Permutation.Gen(M, M^{'})$ //generate a permutation σ so that $M^{'} = M \circ \sigma$
8. $\sigma^1 \leftarrow \$$ //sample a random permutation
9. $\sigma^2 \leftarrow \sigma \circ (\sigma^1)^{-1}$ //composition of σ and inversion of σ^1
10. Outputs σ^1 to S1 and σ^2 to S2;
11. S1:
    (a) Re-randomise the cipher-texts of *blocks*;
    (b) $M^1 = M \circ \sigma^1$; //apply σ^1 to M
    (c) Sends M^1 to S2.
12. S2:
    (a) Re-randomize the cipher-texts of the blocks in M^1;
    (b) $M^2 = M^1 \circ \sigma^2 = M^{'}$; //apply σ^2 to M^1
    (c) Sends $M^{'}$ to S1.

Bibliography

[1] Nick Nguyen. Introducing firefox send, providing free file transfers while keeping your personal information private, 2019. URL https://blog.mozilla.org/blog/2019/03/12/introducing-firefox-send-providing-free-file-transfers-while-keeping-your-personal-information-private/.

[2] DOMO. Data never sleeps 7.0, 2019. URL https://www.domo.com/learn/data-never-sleeps-7.

[3] Robert McMillan and Ryan Knutson. Yahoo triples estimate of breached accounts to 3 billion, 2017. URL https://www.wsj.com/articles/yahoo-triples-estimate-of-breached-accounts-to-3-billion-1507062804.

[4] Carole Cadwalladr and Emma Graham-Harrison. Revealed: 50 million facebook profiles harvested for cambridge analytica in major data breach. *The Guardian*, 17:2018, 2018.

[5] Jason Silverstein. Hundreds of millions of facebook user records were exposed on amazon cloud server, 2019. URL https://www.cbsnews.com/news/millions-facebook-user-records-exposed-amazon-cloud-server/.

[6] Zack Whittaker. Adultfriendfinder network hack exposes 412 million accounts, 2016. URL https://www.zdnet.com/article/adultfriendfinder-network-hack-exposes-secrets-of-412-million-users/.

[7] Jonathan Stempel and Nandita Bose. Target in $39.4 million settlement with banks over data breach, 2015. URL https://www.reuters.com/article/us-target-breach-settlement-idUSKBN0TL20Y20151203.

[8] Cisco. Cisco 2024 data privacy benchmark study, January 2024. URL https://www.cisco.com/c/dam/en_us/about/doing_business/trust-center/docs/cisco-privacy-benchmark-study-2024.pdf.

[9] IBM. Cost of a data breach report 2023, July 2023. URL https://www.ibm.com/reports/data-breach.

[10] Georg Fuchsbauer, David Pointcheval, and Damien Vergnaud. Transferable constant-size fair e-cash. In *International Conference on Cryptology and Network Security*, pages 226–247. Springer, 2009.

[11] DropSecure. Enabling true file transfer security: How dropsecure safeguards your confidential data. Technical report, DropSecure, 2019.

[12] Roger Dingledine, Nick Mathewson, and Paul Syverson. Tor: The second-generation onion router. Technical report, Naval Research Lab Washington DC, 2004.

[13] Paul Rösler, Christian Mainka, and Jörg Schwenk. More is less: on the end-to-end security of group chats in signal, whatsapp, and threema. In *2018 IEEE European Symposium on Security and Privacy (EuroS&P)*, pages 415–429. IEEE, 2018.

[14] SendSafely. Powerful security that's simple to use, 2019. URL https://www.sendsafely.com/howitworks/.

[15] Craig Gentry, Amit Sahai, and Brent Waters. Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based. In *Advances in Cryptology–CRYPTO 2013: 33rd Annual Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2013. Proceedings, Part I*, pages 75–92. Springer, 2013.

[16] Yanjun Shen, Bin Yu, Shangqi Lai, Xingliang Yuan, Shi-Feng Sun, Joseph K Liu, and Surya Nepal. Oblivsend: Secure and ephemeral file sharing services with oblivious expiration control. In *International Conference on Information Security*, pages 269–289. Springer, 2022.

[17] Philip R Zimmermann. *The official PGP user's guide*. MIT press, 1995.

[18] Trevor Perrin and Moxie Marlinspike. The double ratchet algorithm. *GitHub wiki*, 112, 2016.

[19] Tilman Frosch, Christian Mainka, Christoph Bader, Florian Bergsma, Jörg Schwenk, and Thorsten Holz. How secure is textsecure? In *2016 IEEE European Symposium on Security and Privacy (EuroS&P)*, pages 457–472. IEEE, 2016.

[20] Rafail Ostrovsky. Efficient computation on oblivious rams. In *Proceedings of the twenty-second annual ACM symposium on Theory of computing*, pages 514–523, 1990.

[21] Emil Stefanov, Marten Van Dijk, Elaine Shi, Christopher Fletcher, Ling Ren, Xiangyao Yu, and Srinivas Devadas. Path oram: an extremely simple oblivious ram protocol. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, pages 299–310, 2013.

[22] Elaine Shi, T-H Hubert Chan, Emil Stefanov, and Mingfei Li. Oblivious ram with o ((logn) 3) worst-case cost. In *International Conference on The Theory and Application of Cryptology and Information Security*, pages 197–214. Springer, 2011.

[23] Andrew C Yao. Protocols for secure computations. In *23rd annual symposium on foundations of computer science (sfcs 1982)*, pages 160–164. IEEE, 1982.

[24] Craig Gentry. Fully homomorphic encryption using ideal lattices. In *Proceedings of the forty-first annual ACM symposium on Theory of computing*, pages 169–178, 2009.

[25] Andrew Chi-Chih Yao. How to generate and exchange secrets. In *27th Annual Symposium on Foundations of Computer Science (sfcs 1986)*, pages 162–167. IEEE, 1986.

[26] Peter Kairouz, H Brendan McMahan, Brendan Avent, Aurélien Bellet, Mehdi Bennis, Arjun Nitin Bhagoji, Kallista Bonawitz, Zachary Charles, Graham Cormode, Rachel Cummings, et al. Advances and open problems in federated learning. *Foundations and trends® in machine learning*, 14(1–2):1–210, 2021.

[27] Benny Chor, Eyal Kushilevitz, Oded Goldreich, and Madhu Sudan. Private information retrieval. *Journal of the ACM (JACM)*, 45(6):965–981, 1998.

[28] Cynthia Dwork. Differential privacy. In *International colloquium on automata, languages, and programming*, pages 1–12. Springer, 2006.

[29] Shengmin Xu, Guomin Yang, Yi Mu, and Robert H Deng. Secure fine-grained access control and data sharing for dynamic groups in the cloud. *IEEE Transactions on Information Forensics and Security*, 13(8):2101–2113, 2018.

[30] Guy Zyskind, Oz Nathan, et al. Decentralizing privacy: Using blockchain to protect personal data. In *2015 IEEE security and privacy workshops*, pages 180–184. IEEE, 2015.

[31] Yuchen Yang, Longfei Wu, Guisheng Yin, Lijie Li, and Hongbin Zhao. A survey on security and privacy issues in internet-of-things. *IEEE Internet of things Journal*, 4(5):1250–1258, 2017.

[32] Eli Ben Sasson, Alessandro Chiesa, Christina Garman, Matthew Green, Ian Miers, Eran Tromer, and Madars Virza. Zerocash: Decentralized anonymous payments from bitcoin. In *2014 IEEE symposium on security and privacy*, pages 459–474. IEEE, 2014.

[33] Shen Noether, Adam Mackenzie, et al. Ring confidential transactions. *Ledger*, 1: 1–18, 2016.

[34] Chuan Zhang, Xingqi Luo, Jinwen Liang, Ximeng Liu, Liehuang Zhu, and Song Guo. Pota: Privacy-preserving online multi-task assignment with path planning. *IEEE Transactions on Mobile Computing*, 2023.

[35] OpenPGP. Technical standard, 2016. URL https://www.openpgp.org/about/standard/.

[36] Vipul Goyal, Omkant Pandey, Amit Sahai, and Brent Waters. Attribute-based encryption for fine-grained access control of encrypted data. In *Proceedings of the 13th ACM conference on Computer and communications security*, pages 89–98, 2006.

[37] John Bethencourt, Amit Sahai, and Brent Waters. Ciphertext-policy attribute-based encryption. In *2007 IEEE symposium on security and privacy (SP'07)*, pages 321–334. IEEE, 2007.

[38] Rui Chen, Benjamin Fung, and Bipin C Desai. Differentially private trajectory data publication. *arXiv preprint arXiv:1112.2020*, 2011.

[39] Ghada Almashaqbeh and Ravital Solomon. Sok: Privacy-preserving computing in the blockchain era. In *2022 IEEE 7th European Symposium on Security and Privacy (EuroS&P)*, pages 124–139. IEEE, 2022.

[40] Amos Beimel and Yuval Ishai. Information-theoretic private information retrieval: A unified construction. In *Automata, Languages and Programming: 28th International Colloquium, ICALP 2001 Crete, Greece, July 8–12, 2001 Proceedings 28*, pages 912–926. Springer, 2001.

[41] Radu Sion and Bogdan Carbunar. On the computational practicality of private information retrieval. In *Proceedings of the network and distributed systems security symposium*, pages 2006–06. Internet Society Geneva, Switzerland, 2007.

[42] Vitalik Buterin et al. A next-generation smart contract and decentralized application platform. *white paper*, 3:37, 2014.

[43] General Data Protection Regulation. Regulation (eu) 2016/679 of the european parliament and of the council of 27 april 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing directive 95/46. *Official Journal of the European Union (OJ)*, 59(1-88):294, 2016.

[44] Lydia de la Torre. A guide to the california consumer privacy act of 2018. *Available at SSRN 3275571*, 2018.

[45] Alan Rusbridger. The snowden leaks and the public, 2013.

[46] Bruce Schneier. *Data and Goliath: The hidden battles to collect your data and control your world*. WW Norton & Company, New York, NY, USA, 2015.

[47] Jonathan Mayer, Patrick Mutchler, and John C Mitchell. Evaluating the privacy properties of telephone metadata. *Proceedings of the National Academy of Sciences*, 113(20):5536–5541, 2016.

[48] Weikeng Chen and Raluca Ada Popa. Metal: A metadata-hiding file-sharing system. *IACR Cryptol. ePrint Arch.*, 2020:83, 2020.

[49] WhatsApp. Whatsapp encryption overview: Technical white paper. Technical report, WhatsApp, 2017.

[50] Kate Fitzpatrick. Password protect files with digify passkey encryption, 2019. URL https://help.digify.com/en/articles/747136-password-protect-files-with-digify-passkey-encryption.

[51] Dropbox. Dropbox business security: A dropbox whitepaper. Technical report, Dropbox, 2019.

[52] Telegram. What is a secret chat in telegram, 2019. URL https://telegramguide.com/secret-chat-telegram/.

[53] Gmail. Send & open confidential emails, 2019. URL https://support.google.com/mail/answer/7674059.

[54] Proton Technologies AG. Security details, 2019. URL https://protonmail.com/security-details.

[55] Tutanota. Secure emails at the tip of your finger, 2019. URL https://tutanota.com/security.

[56] Inc SSH Communications Security. Sftp – ssh secure file transfer protocol, 2019. URL https://www.ssh.com/ssh/sftp/.

[57] GunPG. Gnupg — the universal crypto engine, 2017. URL https://gnupg.org/software/index.html.

[58] Warren He, Devdatta Akhawe, Sumeet Jain, Elaine Shi, and Dawn Song. Shadowcrypt: Encrypted web applications for everyone. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, pages 1028–1039. ACM, 2014.

[59] 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 *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 1175–1191. ACM, 2017.

[60] Vincent Primault, Vasileios Lampos, Ingemar Cox, and Emiliano De Cristofaro. Privacy-preserving crowd-sourcing of web searches with private data donor. In *The World Wide Web Conference*, pages 1487–1497. ACM, 2019.

[61] Saba Eskandarian, Jonathan Cogan, Sawyer Birnbaum, Peh Chang Wei Brandon, Dillon Franke, Forest Fraser, Gaspar Garcia Jr, Eric Gong, Hung T Nguyen, Taresh K Sethi, et al. Fidelius: Protecting user secrets from compromised browsers. *arXiv preprint arXiv:1809.04774*, 2018.

[62] Helei Cui, Yajin Zhou, Cong Wang, Xinyu Wang, Yuefeng Du, and Qian Wang. Ppsb: An open and flexible platform for privacy-preserving safe browsing. *IEEE Transactions on Dependable and Secure Computing*, 2019.

[63] Raymond Cheng, Fan Zhang, Jernej Kos, Warren He, Nicholas Hynes, Noah Johnson, Ari Juels, Andrew Miller, and Dawn Song. Ekiden: A platform for confidentiality-preserving, trustworthy, and performant smart contracts. In *2019 IEEE European Symposium on Security and Privacy (EuroS&P)*, pages 185–200. IEEE, 2019.

[64] Henry Corrigan-Gibbs, Dan Boneh, and David Mazières. Riposte: An anonymous messaging system handling millions of users. In *2015 IEEE Symposium on Security and Privacy*, pages 321–338, Washington, DC, USA, 2015. IEEE.

[65] David Isaac Wolinsky, Henry Corrigan-Gibbs, Bryan Ford, and Aaron Johnson. Dissent in numbers: Making strong anonymity scale. In *Presented as part of the 10th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 12)*, pages 179–182, Berkeley, CA, USA, 2012. USENIX Association.

[66] Jelle Van Den Hooff, David Lazar, Matei Zaharia, and Nickolai Zeldovich. Vuvuzela: Scalable private messaging resistant to traffic analysis. In *Proceedings of the 25th Symposium on Operating Systems Principles*, pages 137–152, New York, NY, USA, 2015. ACM.

[67] Sebastian Angel and Srinath Setty. Unobservable communication over fully untrusted infrastructure. In *12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16)*, pages 551–569, Berkeley, CA, USA, 2016. USENIX Association.

[68] David Lazar and Nickolai Zeldovich. Alpenhorn: Bootstrapping secure communication without leaking metadata. In *12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16)*, pages 571–586, Berkeley, CA, USA, 2016. USENIX Association.

[69] Nirvan Tyagi, Yossi Gilad, Derek Leung, Matei Zaharia, and Nickolai Zeldovich. Stadium: A distributed metadata-private messaging system. In *Proceedings of the 26th Symposium on Operating Systems Principles*, pages 423–440, New York, NY, USA, 2017. ACM.

[70] Nikita Borisov, George Danezis, and Ian Goldberg. Dp5: A private presence service. *Proceedings on Privacy Enhancing Technologies*, 2015(2):4–24, 2015.

[71] Adi Shamir. How to share a secret. *Communications of the ACM*, 22(11):612–613, 1979.

[72] Changyu Dong, Liqun Chen, and Zikai Wen. When private set intersection meets big data: an efficient and scalable protocol. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, pages 789–800, New York, NY, USA, 2013. ACM.

[73] Burton H Bloom. Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, 13(7):422–426, 1970.

[74] Nick Szabo. Formalizing and securing relationships on public networks, 1997. URL https://ojphi.org/ojs/index.php/fm/article/view/548/469.

[75] Remy Remigius Zgraggen. Smart insurance contracts based on virtual currency: Legal sources and chosen issues. In *Proceedings of the 2019 International Electronics Communication Conference*, pages 99–102, New York, NY, USA, 2019. ACM.

[76] Hokey Min. Blockchain technology for enhancing supply chain resilience. *Business Horizons*, 62(1):35–45, 2019.

[77] Merlinda Andoni, Valentin Robu, David Flynn, Simone Abram, Dale Geach, David Jenkins, Peter McCallum, and Andrew Peacock. Blockchain technology in the energy sector: A systematic review of challenges and opportunities. *Renewable and Sustainable Energy Reviews*, 100:143–174, 2019.

[78] Yuncong Hu, Sam Kumar, and Raluca Ada Popa. Ghostor: Toward a secure data-sharing system from decentralized trust. In *17th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 20)*, pages 851–877, 2020.

[79] Konstantinos Christidis and Michael Devetsikiotis. Blockchains and smart contracts for the internet of things. *Ieee Access*, 4:2292–2303, 2016.

[80] Speedtest. Speedtest global index, 2020. URL https://www.speedtest.net/global-index.

[81] Snapchat. Snapchat support, 2019. URL https://support.snapchat.com/.

[82] Thomas Hardjono and Alex Pentland. Verifiable anonymous identities and access control in permissioned blockchains, 2019. URL https://arxiv.org/abs/1903.04584.

[83] Kaitai Liang, Joseph K. Liu, Rongxing Lu, and Duncan S. Wong. Privacy concerns for photo sharing in online social networks. *IEEE Internet Comput.*, 19(2):58–63, 2015.

[84] Cong Zuo, Jun Shao, Joseph K. Liu, Guiyi Wei, and Yun Ling. Fine-grained two-factor protection mechanism for data sharing in cloud storage. *IEEE Trans. Inf. Forensics Secur.*, 13(1):186–196, 2018.

[85] Oded Goldreich. Towards a theory of software protection and simulation by oblivious rams. In *Proceedings of the nineteenth annual ACM symposium on Theory of computing*, pages 182–194, 1987.

[86] Benny Pinkas and Tzachy Reinman. Oblivious ram revisited. In *Annual cryptology conference*, pages 502–519. Springer, 2010.

[87] Matteo Maffei, Giulio Malavolta, Manuel Reinert, and Dominique Schröder. Privacy and access control for outsourced personal records. In *2015 IEEE Symposium on Security and Privacy*, pages 341–358. IEEE, 2015.

[88] Matteo Maffei, Giulio Malavolta, Manuel Reinert, and Dominique Schröder. Maliciously secure multi-client oram. In *International Conference on Applied Cryptography and Network Security*, pages 645–664. Springer, 2017.

[89] Ariel Hamlin, Rafail Ostrovsky, Mor Weiss, and Daniel Wichs. Private anonymous data access. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 244–273. Springer, 2019.

[90] Michael Backes, Amir Herzberg, Aniket Kate, and Ivan Pryvalov. Anonymous ram. In *European Symposium on Research in Computer Security*, pages 344–362. Springer, 2016.

[91] Wen Liu, Yong-Bin Wang, Zheng-Tao Jiang, and Yi-Zhen Cao. A protocol for the quantum private comparison of equality with χ-type state. *International Journal of Theoretical Physics*, 51(1):69–77, 2012.

[92] Li Shundong, Wu Chunying, Wang Daoshun, and Dai Yiqi. Secure multiparty computation of solid geometric problems and their applications. *Information Sciences*, 282:401–413, 2014.

[93] Tomas Toft. Secure data structures based on multi-party computation. In *Proceedings of the 30th annual ACM SIGACT-SIGOPS symposium on Principles of distributed computing*, pages 291–292, 2011.

[94] Dan Bogdanov, Margus Niitsoo, Tomas Toft, and Jan Willemson. High-performance secure multi-party computation for data mining applications. *International Journal of Information Security*, 11(6):403–418, 2012.

[95] Zhangjie Fu, Kui Ren, Jiangang Shu, Xingming Sun, and Fengxiao Huang. Enabling personalized search over encrypted outsourced data with efficiency improvement. *IEEE transactions on parallel and distributed systems*, 27(9):2546–2559, 2015.

[96] Xiao Wang, Hubert Chan, and Elaine Shi. Circuit oram: On tightness of the goldreich-ostrovsky lower bound. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, pages 850–861, 2015.

[97] Samee Zahur, Xiao Wang, Mariana Raykova, Adrià Gascón, Jack Doerner, David Evans, and Jonathan Katz. Revisiting square-root oram: efficient random access in multi-party computation. In *2016 IEEE Symposium on Security and Privacy (SP)*, pages 218–234. IEEE, 2016.

[98] Jack Doerner and Abhi Shelat. Scaling oram for secure computation. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 523–535, 2017.

[99] Xiao Shaun Wang, Yan Huang, TH Hubert Chan, Abhi Shelat, and Elaine Shi. Scoram: oblivious ram for secure computation. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, pages 191–202, 2014.

[100] Daniel S Roche, Adam Aviv, and Seung Geol Choi. A practical oblivious map data structure with secure deletion and history independence. In *2016 IEEE Symposium on Security and Privacy (SP)*, pages 178–197. IEEE, 2016.

[101] Taher ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE transactions on information theory*, 31(4):469–472, 1985.

[102] Teodor Tomic, Korbinian Schmid, Philipp Lutz, Andreas Domel, Michael Kassecker, Elmar Mair, Iris Lynne Grixa, Felix Ruess, Michael Suppa, and Darius Burschka. Toward a fully autonomous uav: Research platform for indoor and outdoor urban search and rescue. *IEEE robotics & automation magazine*, 19(3): 46–56, 2012.

[103] Federal Aviation Administration. Small Unmanned Aircraft Systems (sUAS) Advisory Circular. Advisory Circular AC 107-2, U.S. Department of Transportation, 2021.

[104] Nathan Michael, Jonathan Fink, and Vijay Kumar. Cooperative manipulation and transportation with aerial robots. *Autonomous Robots*, 30:73–86, 2011.

[105] Department of Defense. Unmanned Systems Integrated Roadmap FY2017-2042. Technical report, Office of the Under Secretary of Defense for Acquisition and Sustainment, 2018. URL https://apps.dtic.mil/sti/pdfs/AD1059546.pdf.

[106] Yingao Elaine Yao, Pritam Dash, and Karthik Pattabiraman. Swarmfuzz: Discovering gps spoofing attacks in drone swarms. In *2023 53rd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pages 366–375. IEEE, 2023.

[107] Azade Fotouhi, Haoran Qiang, Ming Ding, Mahbub Hassan, Lorenzo Galati Giordano, Adrian Garcia-Rodriguez, and Jinhong Yuan. Survey on uav cellular communications: Practical aspects, standardization advancements, regulation, and security challenges. *IEEE Communications surveys & tutorials*, 21(4):3417–3442, 2019.

[108] M ENISA. Good practices for security of internet of things in the context of smart manufacturing, 2018.

[109] Amylia Ait Saadi, Assia Soukane, Yassine Meraihi, Asma Benmessaoud Gabis, Seyedali Mirjalili, and Amar Ramdane-Cherif. Uav path planning using optimization approaches: A survey. *Archives of Computational Methods in Engineering*, 29(6):4233–4284, 2022.

[110] Miguel E Andrés, Nicolás E Bordenabe, Konstantinos Chatzikokolakis, and Catuscia Palamidessi. Geo-indistinguishability: Differential privacy for location-based systems. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, pages 901–914, 2013.

[111] Adam Sealfon. Shortest paths and distances with differential privacy. In *Proceedings of the 35th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, pages 29–41, 2016.

[112] Ugur Ilker Atmaca, Carsten Maple, Gregory Epiphaniou, and Mehrdad Dianati. A privacy-preserving route planning scheme for the internet of vehicles. *Ad Hoc Networks*, 123:102680, 2021.

[113] Martin Florian, Sören Finster, and Ingmar Baumgart. Privacy-preserving cooperative route planning. *IEEE Internet of Things Journal*, 1(6):590–599, 2014.

[114] Farhad Farokhi, Iman Shames, and Karl H Johansson. Private routing and ride-sharing using homomorphic encryption. *IET Cyber-Physical Systems: Theory & Applications*, 5(4):311–320, 2020.

[115] Haining Yu, Lailai Yin, Hongli Zhang, Dongyang Zhan, Jiaxing Qu, and Guangyao Zhang. Road distance computation using homomorphic encryption in road networks. *Computers, Materials & Continua*, 69(3), 2021.

[116] Mohammed Ramadan, Guohong Du, Fagen Li, and Chunxiang Xu. A survey of public key infrastructure-based security for mobile communication systems. *Symmetry*, 8(9):85, 2016.

[117] Yongho Ko, Jiyoon Kim, Daniel Gerbi Duguma, Philip Virgil Astillo, Ilsun You, and Giovanni Pau. Drone secure communication protocol for future sensitive applications in military zone. *Sensors*, 21(6):2057, 2021.

[118] Abdulhadi Shoufan, Hassan AlNoon, and Joonsang Baek. Secure communication in civil drones. In *Information Systems Security and Privacy: First International Conference, ICISSP 2015, Angers, France, February 9-11, 2015, Revised Selected Papers 1*, pages 177–195. Springer, 2015.

[119] Yi Zhou, Cunhua Pan, Phee Lep Yeoh, Kezhi Wang, Maged Elkashlan, Branka Vucetic, and Yonghui Li. Secure communications for uav-enabled mobile edge computing systems. *IEEE Transactions on Communications*, 68(1):376–388, 2019.

[120] Tong Bai, Jingjing Wang, Yong Ren, and Lajos Hanzo. Energy-efficient computation offloading for secure uav-edge-computing systems. *IEEE Transactions on Vehicular Technology*, 68(6):6074–6087, 2019.

[121] Pericle Perazzo, Francesco Betti Sorbelli, Mauro Conti, Gianluca Dini, and Cristina M Pinotti. Drone path planning for secure positioning and secure position verification. *IEEE Transactions on Mobile Computing*, 16(9):2478–2493, 2016.

[122] Yin-Chen Liu, Gianluca Bianchin, and Fabio Pasqualetti. Secure trajectory planning against undetectable spoofing attacks. *Automatica*, 112:108655, 2020.

[123] Aabid Rashid, Diwankshi Sharma, Tufail A Lone, Sumeet Gupta, and Sachin Kumar Gupta. Secure communication in uav assisted hetnets: a proposed model. In *Security, Privacy, and Anonymity in Computation, Communication, and Storage: 12th International Conference, SpaCCS 2019, Atlanta, GA, USA, July 14–17, 2019, Proceedings 12*, pages 427–440. Springer, 2019.

[124] Steven LaValle. Rapidly-exploring random trees: A new tool for path planning. *Research Report 9811*, 1998.

[125] Peter Hart, Nils Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968. doi: 10.1109/tssc.1968.300136. URL https://doi.org/10.1109/tssc.1968.300136.

[126] Edsger W Dijkstra. A note on two problems in connexion with graphs. *Numerische mathematik*, 1(1):269–271, 1959.

[127] Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, and Clifford Stein. *Introduction to algorithms*. MIT press, 2022.

[128] Manuel Suárez-Albela, Tiago M Fernández-Caramés, Paula Fraga-Lamas, and Luis Castedo. A practical performance comparison of ecc and rsa for resource-constrained iot devices. In *2018 Global Internet of Things Summit (GIoTS)*, pages 1–6. IEEE, 2018.

[129] Peter E Hart, Nils J Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968.

[130] Nils J Nilsson. *Principles of artificial intelligence*. Springer Science & Business Media, 1982.

[131] Miguel Castro and Barbara Liskov. Practical byzantine fault tolerance and proactive recovery. *ACM Transactions on Computer Systems (TOCS)*, 20(4):398–461, 2002.

[132] Li Li, Yuxi Fan, Mike Tse, and Kuo-Yi Lin. A review of applications in federated learning. *Computers & Industrial Engineering*, 149:106854, 2020.