CONTRIBUTIONS TO PRIVACY-PRESERVING TECHNOLOGY

JIAZHUO LYU

PhD

The Hong Kong Polytechnic University

2025

The Hong Kong Polytechnic University

Department of Computing

Contributions to Privacy-Preserving Technology

Jiazhuo LYU

A thesis submitted in partial fulfilment of the requirements for the degree of Doctor of Philosophy

January 2024

CERTIFICATE OF ORIGINALITY

I hereby declare that this thesis is my own work and that, to the best of my knowledge and belief, it reproduces no material previously published or written, nor material that has been accepted for the award of any other degree or diploma, except where due acknowledgement has been made in the text.

_____ (Signed)

LYU Jiazhuo _____ (Name of student)

Abstract

In recent decades, preserving privacy has become vital for personal security, social freedom, and economic prosperity. Within this context, secure multi-party computation (MPC) has gained prominence as a key method in the privacy-preserving research field. MPC, a theoretical framework, addresses collaborative computing challenges among distrustful entities without needing a trusted third party. This framework assures both the confidentiality of inputs and the integrity of computations. It employs cryptographic principles to safeguard participant data during computations while ensuring accurate outcomes, all without a trusted third party. In this thesis, we focus on developing efficient MPC protocols with robust functionalities that are adaptable to various scenarios. Specifically, we introduce a novel MPC protocol tailored for applications in e-voting, k-means clustering in machine learning, and fluid participant environments for general computational tasks.

Firstly, we introduce a decentralized e-voting system utilizing smart contract technology. E-voting is a critical application of MPC that significantly impacts social activities. The integrity of voting results and voter privacy are paramount. Our protocol integrates blockchain with smart contract capabilities, linkable ring signatures, and threshold encryption to ensure security and privacy. This design effectively decentralizes trust, ensuring that the voting outcome remains accurate even if some participants are malicious. The system is implemented on an Ethereum private network, offering a robust solution for secure e-voting. Additionally, we provide an analysis of the system's feasibility, including considerations of cost in terms of both financial and time resources.

We also present a novel two-party k-means clustering scheme designed for privacy-preserving collaborative data mining. This field aims to extract useful knowledge from distributed datasets,

owned by multiple entities, without leaking the privacy of the data or the results. An increasing number of companies choose to store and process their data through third-party cloud services. As a result, the need for efficient and secure data mining protocols becomes paramount. Existing approaches in this area, however, suffer from high computational and communication overheads, hindering practical application. Our proposed scheme addresses these challenges by encrypting each party's data once before uploading it to the cloud. Our collaborative clustering protocol for k-means, which prioritizes privacy, is primarily implemented in a cloud environment. This process requires $O(k(m+n))$ interactive sessions involving both parties and the cloud server. Here, m and n represent the respective total record counts from each party. We validate the security of our protocol in both semi-honest and malicious security models, the latter considering a scenario where only one party may be corrupted during centroid recomputation. Comprehensive theoretical and experimental analyses of our protocol are also provided, demonstrating its efficiency and security.

Furthermore, we design a fluid MPC protocol based on SPDZ protocol for general function computation tasks with a small preprocessing computation cost. MPC protocols traditionally require participants to be active throughout the computation process. This requirement can be a significant barrier, especially for complex and resource-intensive tasks. Fluid MPC, a significant advancement presented at Crypto 2021, revolutionizes the traditional framework of secure multi-party computation by introducing a highly adaptable and dynamic system. Unlike conventional MPC protocols, which require a static group of parties to remain consistently engaged throughout the computation process, Fluid MPC allows for a fluid and evolving set of participants. This innovative model is specifically engineered to cater to the variable availability of participants' resources. We extend the Le Mans Fluid MPC protocol, which holds a heavy preprocessing overhead. With the assumption that each computation committee and the transfer order are fixed before the preprocessing stage, the cost of preprocessing is extremely low. In addition, our advanced Fluid MPC protocol stands out by supporting an all-but-one dishonest majority secure model, substantially enhancing the security framework.

Publications arising from the thesis

1. Jiang, Zoe L., Ning Guo, Yabin Jin, Jiazhuo Lv, Yulin Wu, Zechao Liu, Junbin Fang, Siu-Ming Yiu, and Xuan Wang. ”Efficient two-party privacy-preserving collaborative k-means clustering protocol supporting both storage and computation outsourcing.” Information Sciences 518 (2020): 168-180.

2. Jiazhuo Lyu, Zoe L. Jiang, Xuan Wang, Zhenhao Nong, Man Ho Au, and Junbin Fang. ”A secure decentralized trustless E-voting system based on smart contract.” In 2019 18th IEEE International Conference On Trust, Security And Privacy In Computing And Communications/13th IEEE International Conference On Big Data Science And Engineering (TrustCom/BigDataSE), pp. 570-577. IEEE, 2019.

Acknowledgements

First and foremost, I express my deepest gratitude to my parents for their unwavering support and encouragement throughout my life. Over the past 27 years, their love, guidance, protection, and inspiration have been my constant source of strength. I am eternally grateful to them for shaping the person I have become.

I extend my sincere thanks to my supervisor, Prof. Man Ho Au, for his invaluable guidance, support, and patience during my PhD studies. He has always encouraged me to pursue research in my areas of interest and has generously shared his time to discuss each research question in depth. His expertise, vast knowledge, and skills in our research field have been immensely beneficial to me.

I am also deeply thankful to all my friends and the kind individuals who have offered their support and assistance when needed. Their willingness to help has been a great source of comfort and encouragement.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

With the rapid advancement of digitalization and big data analytics, the demand for multi-party data has surged. For example, in fields like personal credit risk assessment, the process necessitates the aggregation and joint analysis of various attributes and characteristics. Private data collected during credit evaluations includes personal identifiers (such as identity, address, and occupation), credit transaction records (from personal loans, credit cards, and guarantees), and indicators of credit status [94]. This information contains nearly all facets of an individual's private life. In this context, the importance of privacy-preserving technology has come to the forefront. It aims to protect individual privacy while enabling the necessary data analysis, addressing the critical challenge of maintaining confidentiality amidst the expanding scope of data collection and analysis.

Secure multi-party computation (MPC) is a cryptographic method enabling multiple parties to collaboratively compute a goal without needing a trusted third party [39]. It ensures that participants cannot access each other's input information except for the final results. This concept was proposed by Academician Andrew Yao in 1982 [89]. Over the years, MPC has evolved into a significant branch of cryptography, offering algorithmic protocols for privacy protection. The implementation of MPC can be divided into the following aspects:

1. General Protocol: This category includes protocols capable of computing any discrete function representable as a fixed-size circuit. Examples include Yao's garbled circuit protocol [8], the GMW (Goldreich-Micali-Wigderson) protocol [40], and the SPDZ protocol. These are versatile and widely applicable in various cryptographic computations.

2. Specific Protocol: In contrast, specific protocols are tailored for particular functions where general protocols may be inefficient due to significant overhead. For functions like Private Set Intersection (PSI) [68], e-voting [68], and e-bidding [5], customized protocols are developed to address the unique requirements and constraints of these applications.

The key protocols in Multi-party Computation (MPC) can be broadly categorized into several distinct types, each with different features and secure models.

1. Secret Sharing (SS) [21]: Secret sharing is a method used to distribute a secret amongst a group of participants, each of whom is allocated a share of the secret. The key idea is that the secret can only be reconstructed when a sufficient number of shares (typically more than a certain threshold) are combined together. When less than this specified number of shares are amalgamated, they disclose no details about the secret. The secret sharing-based MPC protocol usually can be divided into 3 stages:

   (a) Distribution of Secret Shares: Each party's private input is split into secret shares using a secret sharing scheme. These shares are then distributed among all the participating parties.

   (b) Computational Operations on Shares: The participants execute calculations using their respective shares. These operations are designed to replicate those that would have been performed on the original inputs.

   (c) Reconstruction: After the computations, the parties combine their resulting shares to reconstruct the output of the function. Importantly, during this process, the individual inputs of the parties are never reconstructed or revealed.

2. Garbled Circuit (GC) [8]: This technique was introduced by Andrew Yao in the 1980s as part of his solution to the millionaires' problem, where two millionaires want to find out who is richer without revealing their actual wealth. In a garbled circuit, a Boolean circuit (representing a computational task) is turned into a garbled version where the true functionality is obscured. The garble circuit-based MPC protocol usually can be divided into the following stages:

   (a) Circuit Construction: First, the computation to be performed is represented as a Boolean circuit. This circuit consists of gates (like AND, OR, NOT) and wires connecting these gates.

   (b) Garbling the Circuit: One party, often called the garbler, transforms this circuit into a garbled circuit. Each wire in the original circuit is associated with two random keys, designated for 0 and 1 respectively. The garbler then encrypts the output keys of each gate with the input keys in a way that only the correct combination of input keys will decrypt the correct output key.

   (c) Input Encoding: Each party encodes their inputs with the appropriate keys provided by the garbler. In the event that a party's input bit is 1, they utilize the key designated for 1 on the corresponding wire, and conversely, if it's 0, they use the 0 key.

   (d) Circuit Evaluation: The evaluator, who may or may not be the same as the garbler, then processes the garbled circuit. Without knowing the actual inputs or what each gate is doing, the evaluator uses the keys corresponding to their inputs to progressively decrypt the garbled gates and obtain keys for the next level of wires, until the output is reached.

   (e) Output Decryption: Finally, the output keys are translated back into the actual output of the computation.

3. Homomorphic Encryption (HE) [22]: In the context of MPC, homomorphic encryption is used to ensure that the inputs of each party remain private, even as they are being used to

compute some joint function. The process of this kind of protocol is quite straightforward. Firstly, each party encrypts its input using a homomorphic encryption scheme. Then, the computing entity performs the desired computations directly on the encrypted data and gets the encrypted result. Finally, the relevant party or parties can then decrypt the result to obtain the final plaintext output.

4. Oblivious Transfer (OT) [43]: Oblivious transfer is a foundational primitive in many MPC protocols, especially those designed for two-party or small-number-party computations. It is used as a building block to achieve secure computation, ensuring that parties can jointly compute a function over their inputs while keeping those inputs private.

These technologies allow for the utilization of data without exposing the original content, thus safeguarding privacy.

Multi-party computation (MPC) finds wide-ranging applications in areas like multi-party joint data analysis, which encompasses Private Information Retrieval (PIR) [18], Private Set Intersection (PSI) [68], and trusted data exchanges. Additionally, specific applications such as secure e-voting and e-bidding represent specialized forms of MPC protocols. The development and availability of several open-source libraries, such as ABY [27], EMP-toolkit [11], FRESCO [81], JIFF [53], MP-SPDZ [49], MPyC [79], SCALE-MAMBA [4], and TinyGarble [83], have significantly contributed to the practical deployment and broader application of MPC technologies.

1.1 Thesis Outline

The rest of this thesis is organized as follows:

- Chapter 2 lays the foundational elements for the content that follows, including a range of essential notations and definitions.

- Chapter 3 provides the e-voting protocol based on Ethereum. The discussion of the correctness of the protocol and the security analysis are also proposed.
- Chapter 4 presents the privacy-preserving collaborative k-means clustering protocol. Additionally, it explores the protocol's efficiency from both theoretical and experimental perspectives.
- Chapter 5 presents a fluid MPC protocol with a small preprocessing overhead.
- Chapter 6 offers concluding remarks and potential directions for future research.

1.2 A Secure E-voting System Based on Blockchain

E-voting is widely used in social life. However, it is not obvious how to ensure the outcome is respected when the decision is financially or politically related. The correctness, security, and privacy are always the most important characteristics. Secure e-voting is a kind of secure multi-party computation [39]. In the voting process, a set of people make their choices, and their choices can be kept secret. The majority of electronic voting systems require a reliable public bulletin board to ensure a uniform perspective for all voters. However, it is not clear to the election administrator that the public bulletin board can be completely trusted. Some people realize blockchain can be used as a bulletin board because the content is publicly trusted.

Blockchain [69] served as a decentralized database that provides new tools for creating a trustless and decentralized system. In the blockchain system, there is no trusted centralized coordinator. Instead, each node that is involved in the blockchain system holds the data block locally. Blockchain technology is upheld by a peer-to-peer network that is decentralized and allows open membership. At first, this technology is designed for money transfer. With the development of it, researchers are trying to reuse Blockchain in other research areas such as coordinating the Internet of Things [70], carbon dating [20] and health-care [33]. This sparked

the invention of Ethereum [87], which is well known as a milestone in the development of blockchain. It owns a Turing complete programming language, and users can realize the function by the smart contract in the Ethereum network.

Blockchain technology has the potential to serve as a trusted public bulletin board in voting systems. In addition, the smart contract on the blockchain serves as a trusted computer whose result is publicly trusted. However, replacing the bulletin board with blockchain is not a good idea. Because there will be too many transactions for voters to discern and the computation on the blockchain is very hard, this could be seen in [93].

In this paper, we propose a decentralized, trustless e-voting system based on blockchain. The decentralized system means the computation is dependent on a decentralized blockchain. The trustless system means we do not need to rely on the election administrator; the trust is separated from all voters. The correctness of the system depends on the whole protocol. In addition, all voters can have cryptographic assurance that the privacy of each voter can be protected.

To ensure that nobody can tally the election result before the end of the election, the scheme uses threshold encryption without a trusted third party [28, 80]. In addition, even if the election administrator is malicious, the tally result will not be changed. The encryption method is to set up a pair of public-secret keys. The public key is known to all parties, while the secret key is separated to all parties, and nobody gets the complete secret key before the key reconstruction stage. When at least t of n parties upload their secrets, the secret key is reconstructed.

In order to identify the anonymous signature, we use the linkable ring signature [6, 63, 64]. A linkable ring signature allows a member to generate a signature from a list of public keys and a secret key whose corresponding public key is in the list. But nobody (except the generator) could know who generated the signature. It makes a participant to be anonymous during the voting process. The more users involved in the signature, the more anonymous it could be. The public checker could verify that whether two signatures on different messages are generated by

the same signer.

The voting protocol is deployed on Ethereum by the smart contract. The Ethereum script allows users to write the required smart contracts on Ethereum and implement powerful functions through smart contracts to implement decentralized applications. All nodes of the Ethereum network run the contract code independently to ensure the credibility of the final result, which is publicly verifiable.

1.2.1 Related Work

The e-voting system was first raised by Chaum in 1981 [17]. From then on, people focused on the e-voting system. According to cryptographic technologies, people divided the protocol into three kinds:

- **Mixed-network**: E-voting system based on mixed-network was first proposed by Chaum [17]. The basic principle is that multiple input signals are confusing through the mixed-network, and then output multiple signals are cut off the association with the sender. However, the implementation of the mixed-network requires a large amount of zero-knowledge proof [35] to ensure that the servers participating in the hybrid computing have not tampered with the votes.

- **Blind/Ring signature**: Chaum first introduced the concept of a blind signature in 1983 [15]. Unlike in a standard public key signature, where the signer is aware of the content being signed, the blind signature approach differs. However, in the process of blind signature, the signer does not know the content of the file that he signed. When the file is revealed, the signer can verify his signature and get the content of the file, but he does not know the time of the signature generation and who sent the file. Legal voters can not verify that their voting content is properly counted, nor can they verify the correctness

of the counting process. The FOO protocol [36] is representative of the blind signature electronic voting scheme. The protocol uses the blind signature to ensure the uniqueness and privacy of the ballot, and the fairness of voting is achieved through bit commitment. The FOO protocol is the first electronic voting solution that truly meets the basic security needs of electronic voting, pushing electronic voting from the theoretical stage to the practical stage. The paper referenced as [19] marks the initial application of a linkable ring signature in an electronic voting system. These are the fundamentals of a linkable ring signature-based e-voting system.

- **Homomorphic encryption**: E-voting system based on homomorphic encryption of Elgamal [9, 51] was first proposed in 1997. Homomorphic features allow one to operate on ciphertext without decrypting them. In the tally process, the ballots do not require a decryption operation. This feature can greatly improve the privacy of the ballot and the anonymity of the voting [23, 48]. The privacy of the ballots and the anonymity of the voters depend on the security of the homomorphic encryption algorithm.

However, such voting protocols [2] need a centralized trusted party to control the voting process. The blockchain technique and smart contracts provide new ideas for e-voting. Zhao proposed a voting agreement in 2015 [93], which introduced a punish/reward scheme for voters' illegal or legal behaviors. Though the protocol is hard to carry out in the real world because each voter needs to set up a lot of transactions, it is the first attempt to utilize blockchain to solve the voting problem. In 2017, McCorry proposed an electronic voting protocol based on smart contracts [66], using a homomorphic encryption scheme. However, if voters give up voting, the whole protocol needs to be re-run. Bin proposes a practical voting system that is platform-independent, secure, and verifiable [90]. The system is based on smart contracts on Blockchain. In the system, the public key is mastered by the election administrator. Once the administrator is malicious, the voting will be destroyed. This problem also happens to [52].

1.3 Efficient Two-party Privacy-preserving Collaborative k-means Clustering Protocol

Collaborative data mining aims to address how we can tackle the challenge of using data mining methods on scattered data to extract knowledge, which is one of the most important ways to build robust models. However, such collaboration may not be easily achieved due to privacy concerns. For example, in the US, medical data release is not allowed before the de-identification process, as claimed in the Health Insurance Portability and Accountability Act (HIPPA). In the European Union, it has enforced many terms to protect user privacy and prohibit direct data sharing among institutions. Such enforcement creates a substantial barrier for researchers to execute collaborative data mining and further benefits from data sharing. In terms of the privacy concern of collaborative data mining, the idea of privacy-preserving data mining is proposed [92]. Nowadays, there are two main techniques to achieve privacy: differential privacy and homomorphic encryption.

Differential privacy, where the rigorous definition was proposed in [32], has the advantage of efficiency while it may lose accuracy. It has been extended to various applications, such as Naive Bayes [58] and deep learning [1]. Homomorphic encryption is a kind of encryption that enables computation on encrypted data. It can provide accurate computation with the sacrifice of efficiency. Moreover, only a little fully homomorphic encryption can support all kinds of computation on ciphertext [29, 95].

With the advent of cloud computing, end-users outsource their data to cloud services to perform data mining, which is called both data storage and computation outsourcing. In such new infrastructure, to incentivize end-users to join in collaborative mining, privacy becomes one of the most important obstacles [61].

Clustering is designed to group a set of objects into clusters according to some kind of mea-

surement, such that objects within a cluster are similar while dissimilar to those in other clusters. It has been widely used in the applications of medicine, banking, etc. In terms of privacy, many pieces of research work have been launched to study privacy-preserving clustering protocols. However, most of them assume that data is centralized. In this paper, a privacy-preserving collaborative clustering protocol supporting both storage and computation outsourcing will be proposed.

1.3.1 Related Work

The first piece of work on privacy-preserving data mining was given by [3, 60] for the ID3 decision trees classification on horizontally partitioned data using different models of privacy. Lindell's work [60] allows two-party to compute a decision tree based on the combined set of data without revealing each other's data records. Agrawal [3] developed a method allowing one party to delegate data mining tasks to another party without disclosing private data.

Vaidya and Clifton [85] were the pioneers in introducing a multi-party privacy-preserving k-means clustering protocol for vertically partitioned data. Their protocol maintains the confidentiality of each party's data through secure permutation and homomorphic encryption, enabling secure computation and comparison of distances. Jha et al. [46] proposed two privacy-preserving protocols for two-party weighted average calculations, one based on oblivious polynomial evaluation and the other on homomorphic encryption. Their homomorphic encryption experiment successfully clustered a dataset with 5,687 samples and 12 features in about 66 seconds. Jagannathan and Wright [45] expanded this concept to arbitrarily partitioned data, a broader category encompassing both horizontal and vertical partitions.

Bunn and Ostrovsky [14] introduced an efficient two-party k-means clustering protocol for arbitrarily partitioned data, preserving privacy without disclosing any intermediate values using division and random value protocols. Doganay et al. [31] suggested a novel privacy-preserving

k-means clustering protocol, but it depended on a trusted third party for privacy assurance. Patel et al. [72, 73] later offered various schemes in the malicious model, though these were not particularly efficient.

Liu et al. [62] devised a one-party privacy-preserving k-means clustering protocol, enabling users to outsource storage and computation to the cloud without revealing data or mining results to the cloud or other parties. They extended this framework in [65] to include two parties and the cloud, although this increased the computational and interactive costs. Li et al. [59] introduced a privacy-preserving C4.5 decision tree algorithm for horizontally and vertically partitioned datasets. In [57], Li et al. proposed a method for a classifier owner to delegate privacy-preserving classification services to a remote server, including two secure classification protocols for the Naive Bayes classifier. Several protocols involve a trusted third party for authorization, a common practice in outsourced storage systems [54, 58]. Other data mining protocols are discussed in [55, 56, 88], with a detailed comparison presented in Table 1.1.

Table 1.1: Comparison of existing privacy-preserving data mining protocols

Paper	Supported algorithm	Partition model	Security model	Parties	Cloud	Cryptographic techniques
LP00 [60]	ID3	Horizontal	Semi-honest	2	0	Oblivious transfer Randomizing function
AS00 [3]	ID3	Horizontal	Semi-honest	1	1	Oblivious circuit evaluation A protocol for computing $x\ln x$
VC03 [85]	k-means	Vertical	Semi-honest	> 2	0	Paillier encryption Yao's evaluation circuit
JW05 [45]	k-means	Arbitrary	Semi-honest	2	0	Random shares Yao's evaluation circuit
BO07 [14]	k-means	Arbitrary	Semi-honest	2	0	Paillier encryption Secure scalar product
SK10 [78]	k-means	Arbitrary	Semi-honest	> 1	0	Random shares Yao's evaluation circuit
DPS08 [31]	k-means	Vertical	Semi-honest	> 3	0	Additive secret sharing
UMS10 [84]	k-means	Arbitrary	Semi-honest	> 1	> 2	Secret sharing
PPJ13 [72]	k-means	Horizontal	**Malicious**	> 1	0	Shamir's secret sharing Code-based ZK identification
LBY13 [62]	k-means	Horizontal	Semi-honest	1	1	Homomorphic encryption
PSJ14 [73]	k-means	Horizontal or vertical	**Malicious**	> 1	0	Verifiable secret sharing Homomorphic commitments
LJY15 [65]	k-means	Horizontal	Semi-honest	2	1	Liu's encryption Paillier encryption, PPWAP
SRB14 [75]	k-means	Horizontal	Semi-honest	> 1	2	Secure minimum out of k
MR20 [67]	k-means	Horizontal	**Malicious**	> 1	2	Shamir's secret sharing Yao's evaluation circuit Oblivious transfer
BCE21 [13]	k-means	Horizontal	**Malicious**	> 1	2	Secure 2-party computation

1.4 Fluid MPC

Secure multi-party computation (MPC) represents a transformative approach in the realm of privacy-preserving data analysis, allowing multiple parties to jointly compute a function based on their inputs while safeguarding their privacy. Within a MPC protocol, the sole information disclosed about the inputs is that which is deducible from the output of the function. This technology is applicable in various scenarios, such as secure data aggregation, confidential training or evaluation of machine learning models, and threshold cryptography.

The core principle of MPC is to allow computation over distributed data without compromising the privacy of each party's data. This makes it invaluable in situations where sharing raw data is either impractical or forbidden due to privacy concerns. In healthcare, for instance, MPC can enable hospitals to collaborate on patient data for research without violating confidentiality agreements. In finance, it allows for secure risk analysis and fraud detection across multiple institutions without exposing sensitive information.

Traditional MPC protocols assume a fixed group of participants throughout the computation, which limits their applicability in dynamic, real-world scenarios. This static approach struggles to accommodate situations where participants' availability may change, such as long-running computations or collaborative tasks across distributed networks. To overcome these challenges, Fluid MPC was introduced with a clear motivation: to provide a flexible framework where participants can seamlessly join or leave the computation process without disrupting its integrity. This adaptability makes Fluid MPC particularly suited for environments requiring robustness against participant churn, such as large-scale collaborative computations, decentralized applications, and cloud-based services.

Despite the advantages, these fluid models come with their own set of challenges, primarily concerning increased overheads in communication and computation. In models with maximum fluidity, every change in the participants' roster can necessitate additional rounds of communi-

cation and recalculations, potentially leading to inefficiencies. This trade-off between flexibility and overhead is a crucial consideration in the practical application of these protocols. Balancing these aspects is key to optimizing MPC for real-world use, ensuring that it remains both flexible and efficient.

1.4.1 Related Work

Over the last decade, Multi-party Computation (MPC) has evolved from being predominantly theoretical to a practical tool, enabling a group of participants to collaboratively compute a function using their private inputs while maintaining confidentiality. This transformation is largely attributed to the emergence of compilers that convert high-level programming into secure operations like branching, addition, and multiplication on confidential data [30,37]. Compilers such as Sharemind [12], the architecture proposed by Keller et al. [50], ABY [27], and Obliv-C [91] have played a pivotal role in this advancement.

Arithmetic circuits, either operating over integers or modulo p, are preferred in a multitude of applications for their simplicity in representation compared to binary circuit-based bitwise operations. This preference is particularly noticeable in applications like linear programming for satellite collision analysis, where fixed and floating-point computations are extensively utilized [24, 47]. Recent research has also explored reducing storage requirements in sequential computations across different MPC frameworks, incorporating symmetric key algorithms represented as arithmetic circuits [41, 77].

In implementing MPC, one has to choose between two primary approaches: the use of garbled circuits [42, 76, 86] or secret sharing techniques [10, 25, 26]. This paper focuses on the latter, especially given its aptness for evaluating arithmetic circuits, although recent theoretical advances in garbled circuits modulo p by Ball et al. [7] are noteworthy. Our objective is to explore secure computations in a scalable system with numerous participants, ensuring robust

protection against malicious entities, using the SPDZ protocol.

Recent advancements in MPC have seen the development of more practical approaches, notably Fluid MPC [7] and YOSO [38]. These innovative models introduce protocols that accommodate a fluidly changing group of participants. They allow parties to freely join or exit the computational process without disrupting ongoing protocols. This flexibility is particularly advantageous for extensive, prolonged computational tasks, such as intricate scientific research akin to Folding@home projects. In scenarios of maximal fluidity, this concept is taken to an extreme, allowing each participant to be involved for only a single round, thus maximizing the adaptability of potential contributors.

The YOSO (you only speak once) approach [38] extends this idea of maximally fluid MPC protocols, introducing unique variations in its model. It diverges from Fluid MPC by examining how roles are assigned within the protocol. Their solution utilizes blockchain technology for the random selection of a committee for each round. In this system, a committee member's identity remains undisclosed until their contribution has been made, significantly enhancing security by keeping the participants' identities concealed from potential adversaries until their role is complete.

These methodologies both provide information-theoretically secure protocols in an honest majority environment, wherein a majority of participants in any given round are presumed honest. Fluid MPC is engineered to safeguard against abrupt terminations, effectively thwarting attempts by malicious parties to end the protocol prematurely. In contrast, YOSO provides a more robust assurance of guaranteed output delivery, although this heightened security comes with a trade-off in terms of reduced efficiency.

In contrast, the study by Rachuri et al. [74] explores MPC in environments with a dishonest majority. This approach demands only one honest participant per round, offering a more robust security framework, albeit with greater complexity than the honest majority scenario. We will

further discuss our contributions and provide some technical insights in the following sections.

1.5 Connections Between the Three Components

This paper presents a cohesive framework that addresses various challenges in privacy-preserving computation through three interconnected components. Each component builds upon the others to provide a comprehensive solution for secure and efficient computation in diverse scenarios.

First, we introduce a secure electronic voting system tailored for small-scale use cases where strong verifiability is a critical requirement. In such scenarios, it is essential to ensure that every vote is counted correctly while preserving voter anonymity and protecting against tampering. By leveraging cryptographic techniques such as blind signatures and utilizing blockchain as a trusted public bulletin board, the proposed system provides robust guarantees of transparency and integrity. This solution is particularly suitable for applications where the correctness of each individual transaction is paramount, such as board elections or small-scale community voting.

Second, moving to large-scale applications, such as national elections or large-scale surveys, statistical analysis becomes crucial. We focus on the privacy-preserving computation of k-means clustering for analyzing aggregated voting data. In these cases, data from multiple parties must be processed collectively to extract meaningful patterns while ensuring that sensitive information remains confidential. To address this, we propose a secure k-means clustering protocol that outsources the computational workload to a cloud server. The protocol is designed for horizontally partitioned datasets and incorporates homomorphic encryption and secure multiparty computation (MPC) techniques to protect individual data while enabling collaborative analysis. This component bridges the gap between data privacy and the computational demands of large-scale systems.

However, outsourcing computation entirely to external servers introduces additional risks

and limitations. Third, we address two critical challenges of outsourced computation: (1) the potential for malicious behavior by all outsourced servers, which could lead to protocol termination and privacy leakage, and (2) the possibility of computational resource shortages on the servers, hindering efficient execution. To mitigate these issues, we enhance the existing Fluid MPC protocol and propose an optimized Dynamic SPDZ protocol. This improved protocol ensures robustness even in adversarial settings and adapts efficiently to resource constraints. By reducing both computational and communication overheads, the Dynamic SPDZ protocol achieves superior performance, making it well-suited for dynamic, resource-constrained, and adversarial environments.

Together, these three components form a comprehensive framework addressing the key challenges in privacy-preserving computation. The secure electronic voting system lays the foundation for strong verifiability and transparency in small-scale use cases. The k-means clustering protocol extends the framework to large-scale statistical analysis while preserving data privacy. Finally, the Dynamic SPDZ protocol ensures the robustness and efficiency of outsourced computation, providing a practical solution for dynamic and adversarial scenarios. This integrated approach demonstrates how diverse privacy-preserving techniques can be combined to solve real-world problems across different scales and applications.

Chapter 2

Preliminaries

In this section, we give a brief introduction to Ethereum, which is the first blockchain to support smart contracts, linkable ring signatures, and the threshold encryption system used in the e-voting protocol. In addition, we also review homomorphic encryption, some related cryptographic primitives, and the concept of a horizontal data partition. Finally, we introduce the oblivious linear evaluation and universal composed secure model.

2.1 Blockchain and Ethereum

Blockchain is a revolutionary technology that has garnered widespread attention for its potential to transform various industries. At its core, blockchain is a type of distributed ledger technology (DLT) that records transactions in a secure, transparent, and immutable manner. It consists of a series of data blocks, each containing a list of transactions. These blocks are linked and secured using cryptographic principles, forming a chain.

The key features of blockchain include decentralization, transparency, and immutability. Unlike traditional systems, where a single entity controls the database, a blockchain is decentralized and maintained by a network of nodes (computers), with each node holding a copy of

the ledger. This structure ensures transparency, as transactions on the blockchain are visible to all participants, fostering trust in the system. Additionally, once recorded, the data in any given block cannot be altered retroactively without altering all subsequent blocks, which requires the consensus of the network majority.

In terms of operation, when a transaction occurs, it is broadcast to the network and validated by nodes through a consensus process. Once a transaction is validated, it is grouped with other transactions to create a new block of data for the ledger. This block is then added to the existing blockchain in a way that is permanent and unchangeable.

Blockchain technology has applications across numerous fields, including finance (with cryptocurrencies like Bitcoin and Ethereum being the most notable examples), supply chain management, healthcare, and more. Its ability to provide secure, transparent, and efficient transactions makes it a promising technology for the future. Ethereum is a state machine based on orderly transactions. It depends on a distributed P2P computer network so that all the transactions are broadcasted into the network. Ethereum features two distinct account types:

- Externally Owned Account: Controlled by a user through a public/secret key pair. The user is responsible for initiating transactions within the network.
- Contract Account: Governed by the smart contract's code. An externally owned account deploys the smart contract onto the blockchain.

Each externally owned account is associated with a pair of keys: a secret key and its corresponding public key. The secret key is employed for signing transactions, while the public key is utilized to confirm the authenticity of the signature. In contrast, a contract account does not have any private key. It stores the code of a smart contract that decides the flow of the ethers in the account. Both accounts can store and spend a given number of Ethereum native tokens called Ether. Ether is the token to pay for using network computing resources and transactions inside the Ethereum network. The smart contract can't execute code by itself. It must interact

with an owned account to execute the function. Any externally owned accounts can send a transaction to the contract address.

The structure of an Ethereum transaction is:

- From: A signature from an externally owned account address to authorize the transaction.
- To: The receiver's address(externally owned account or contract address).
- Value: Amount of transfer ether.
- Data: Contract code used to create a new contract or execute instructions for the contract.
- Gas price: The price of each unit of gas.
- Total Gas: The maximum amount of gas that the user is willing to pay for the contract.

The Ethereum blockchain can be considered a state machine. Every change of state will cost ether. Each block has a set of transactions. In particular, smart contracts are coding contracts on the blockchain that automatically move digital assets according to predetermined rules. Participants who do not trust each other are allowed to transact safely under the contract without being affected by the third party.

Currently, each transaction in Ethereum must be mined into a block by the winner who wins in the Proof-of-Work scheme [82]. This provides us with a decentralized computing environment. Ethereum offers a public bulletin board and an authenticated broadcast channel, both essential for decentralized internet voting protocols to facilitate coordination among voters effectively. What's more, almost all calculations made during the voting period are public and can be written as smart contracts. Crucially, the security of the entire voting protocol's execution is ensured by the blockchain's consensus mechanism. This establishes a trusted computing environment.

2.2 Linkable Ring Signature

A linkable ring signature is a kind of digital signature that each signer could be anonymous, and only a registered signer could create the signature. In 1991, Chaum and Heyst introduced group signature [16]. A group is a set of users who have different pairs of public/secret keys. In the group, there is a manager who manages all users in the group. For the group signature, we need to trust the group manager. In order to solve this problem, Joseph, Victor, and Duncan formalized a linkable ring signature, which produced a scheme without privacy revocation. The linkable ring signature scheme satisfies three properties: (1) Anonymity: Anybody could not know who generated the signature. (2) Linkability: It is possible to identify when two signatures are produced by the same signer. (3) Spontaneity: There is not a group manager who controls some secret

The scheme being used takes the DLP and is provable under the random oracle model, using a cryptographic hash function as a random function. We adapted this scheme for use over elliptic curves by hashing it into an elliptic curve.

In our implementation, we assume that F_q is a finite cyclic group whose order is a prime number q. $E(F_q)$ is an elliptic curve over the finite group F_q. G is a base point of the curve $E(F_q)$. l is the order of the base point G. Let H_1 be a cryptographic hash function that can map a number into the finite cycle group F_q. Let H_2 be a cryptographic hash function that can map an input to a point of an elliptic curve [44].

We assume there are n users in our group, and each user has their corresponding private key sk_i and their public key $pk_i = sk_iG$. L donates all public keys $L = \{pk_1, pk_2, ..., pk_n\}$

Signature Generation: The user wants to sign message $m \in \{0,1\}^*$ with the secret key x_i.

1. Compute $M = H_2(L)$ and $K = x_iH$.

2. Choose random $c \in F_q$ and compute

$$u_{i+1} = H_1(L, K, m, cG, cH).$$

3. For $j = i+1, .., n, 1, ..., i-1$, choose random $v_j \in F_q$ and compute

$$u_{i+1} = H_1(L, K, m, v_jG + pk_ju_j, v_jM + u_j)$$

4. Compute $v_i = c - sk_iu_i$ mod q

The linkable ring signature is $(u_1, v_1, ..., v_n, K)$

Signature Verification: Any public checker checks $sig(m) = (u_1, v_1, ..., v_n, K)$. m donates the message and L donates all public keys:

1. Compute $M = H_2(L)$

2. For $i \in [0, n]$, compute:

$$\alpha_i = v_iG + u_iy_i$$

$$\beta_i = v_iH + u_iK$$

$$u_{i+1} = H_1(L, K, m, \alpha_i, \beta_i)$$

3. Check whether $u_1 = H_1(L, K, m, \alpha_n, \beta_n)$. If yes, accept. Otherwise, reject.

Linkability: For the same public key list L, given two signature associating with L, $sig(m_1) = (u_1, v_1, ..., v_n, K)$ $sig(m^{'}) = (u_1^{'}, v_1^{'}, ..., v_n^{'}, K^{'})$. And m and $m^{'}$ could be two different messages. Any public checker verifies whether $K = K^{'}$. If $K = K'$, the two signatures on different messages are generated by the same user. Otherwise, they are generated by different users.

In a voting system, ring signatures play a crucial role in ensuring voter anonymity and unlinkability. By allowing a voter to generate a signature that appears indistinguishable from those of other members in a predefined group, ring signatures prevent the identification of the actual signer. This guarantees that the identity of the voter cannot be traced back to their vote, providing strong privacy without requiring a trusted third party for anonymity.

2.3 Threshold Encryption Without Trusted Third Party

In the threshold encryption system, every member of the encryption group shares a pair of public/secret keys. The public key is known by all voters, while the secret key is separated from all voters without a trusted third party. In addition, only when some of the voters(exceed the threshold) cooperate can the secret key be restructured. In this section, we will mainly show how to distribute the secret key without a trusted third party.

There are n numbers in a group $\{P_i | i \in [1, n]\}$, F_p stands for the finite cycle group whose order is p, and the generator of the group donates g. k is the number of threshold which means the minimum numbers to upload their secret key.

1. P_i chooses $x_i \in F_p$ at random and computes $h_i = g^{x_i}$ The public key h is the sum of all h_i.

2. P_i randomly choose a polynomial $f_i(c) \in Z_q(c)$ of degree at most $k-1$ and $f_i(0) = x_i$

$f_i(c) = f_i + f_{i,1}c + ... + f_{i,k-1}c^{k-1}$

P_i computes $F_{ij} = g^{f_{ij}}$ for $j = 0, ..., k-1$ and publishes these values.

3. When everybody have published these k values, P_i sends $s_{ij} = f_i j$ secretly to P_j for $j = 1, ...n$

4. P_i verifies the data s_{ij} received from P_j. To make sure whether it is consistent with the

previously published values, P_i computes that $g^{s_{ij}} = \sum_{l=0}^{k-1} F_{jl}^{i^I}$. If this fails, stop.

5. P_i computes his share of x (donates s_i) as the sum of all shares received before. Let f be the following polynomial $f(c) = f_1(c) + ... + f_n(c)$. By construction $s_i = f(i)$ and thus, s_i is a share of $f(0) = x$ so that they could restructure the secret key x easily, which could be found in [80].

Threshold encryption without a trusted third party ensures that sensitive operations, such as decrypting a result, require the cooperation of multiple parties, thus distributing trust among them. This approach eliminates the need for a single point of failure or reliance on a trusted entity, enhancing the security and robustness of the system. In the context of secure computation or voting, threshold encryption enables joint decryption only when a predefined number of participants agree, safeguarding against both insider threats and external attacks.

2.4 Homomorphic Encryption

The homomorphic encryption we use is Paillier encryption [71], which is a probabilistic asymmetric 3-tuple encryption algorithm denoted by $\mathsf{Enc}_{P_a} = \{\mathsf{K}, \mathsf{E}, \mathsf{D}\}$.

- $\mathsf{K}(1^\kappa) \rightarrow (pk, sk)$:

  (1) Choose two large prime numbers p and q which satisfy that $\gcd(pq, (p-1)(q-1)) = 1$.

  (2) Calculate $n = pq$ and $\lambda = \mathsf{lcm}(p-1, q-1)$.

  (3) Randomly choose an integer $g \in Z_{n^2}$.

  (4) Check whether there exists $u = (\mathsf{L}(g^\lambda \bmod n^2))^{-1} \bmod n$ where function $\mathsf{L}(\mu) = (\mu - 1)/n$. Then pk is (n, g) and sk is (λ, μ).

- $E_{pk}(x, r) \rightarrow c$:

  Select a random $r \in Z_n^*$ for the message x and the ciphertext is $c = g^x r^n \bmod n^2$.

- $\mathsf{D}_{sk}(c) \rightarrow x$:

  Decrypt the message by $x = \mathsf{L}(c^{\lambda} \bmod n^2)\mu \bmod n$.

In the case of no ambiguity, we remove the subscripts pk of E_{pk} and sk of D_{sk}. Then, the additive homomorphic properties of Paillier encryption are:

$$E(x)E(y) = E(x + y), E(x)^y = E(xy).$$

Homomorphic encryption is a critical component of our secure k-means clustering protocol, enabling computations on encrypted data without revealing the underlying plaintext. This ensures that the privacy of each party's data is preserved throughout the clustering process. Specifically, addition and multiplication operations on ciphertext are utilized to compute distances, compare values, and update centroids in a secure manner. By leveraging the additive homomorphic property of Paillier encryption, our protocol guarantees data confidentiality while maintaining the accuracy of the clustering results.

2.4.1 Basic Secure Computation Primitives

In this section, we review a group of cryptographic primitives that will be used or adapted as toolkits [34] for the proposed protocol. Paillier's public key pk will be known to the public, and the corresponding secret key sk will only be known by P.

(1) Secure Multiplication (SM) Protocol (**Protocol** 1):

With inputs $(E(x), E(y))$, this protocol computes the output $E(xy)$ for C with the help of P. The public key is pk, and the secret key is sk generated by Paillier encryption.

Protocol 1: $SM(E(x), E(y)) \rightarrow E(xy)$

Require: C has $E(x)$ and $E(y)$; P has sk

1. C:
   (a) Pick any two different numbers $r_x, r_y \in Z_N$
   (b) $x^{'} \leftarrow E(x)E(r_x)$, $y^{'} \leftarrow E(y)E(r_y)$
   (c) Send $x^{'}, y^{'}$ to P
2. P:
   (a) $h_x \leftarrow D(x^{'})$, $h_y \leftarrow D(y^{'})$, $h \leftarrow h_x h_y \bmod N$, $h^{'} \leftarrow E(h)$
   (b) Send $h^{'}$ to C
3. C:
   (a) $s \leftarrow h^{'}E(x)^{N-r_y}$, $s^{'} \leftarrow sE(y)^{N-r_x}$
   (b) $E(xy) \leftarrow s^{'}E(r_x r_y)^{N-1}$

(2) Secure Squared Euclidean Distance (SSED) Protocol (**Protocol** 2):

Let $X = (x_1, \cdots, x_\ell)$ and $Y = (y_1, \cdots, y_\ell)$ denote the two ℓ-dimensional vectors, and $[X] = (E(x_1), \cdots, E(x_\ell))$ and $[Y] = (E(y_1), \cdots, E(y_\ell))$ denote the sets of the encrypted components of X and Y. C is with input $([X], [Y])$, and P calculates the corresponding encryption value of the squared Euclidean distance. At the end of the protocol, the final output $E(|X - Y|^2) = \prod_{i=1}^{\ell} E_{pk}((x_i - y_i)^2)$ is known only to C.

Protocol 2: $\mathsf{SSED}([X], [Y]) \rightarrow E_{pk}(|X - Y|^2)$

Require: C has $[X]$ and $[Y]$; P has sk

1. C:
   for $1 \leq i \leq \ell$ **do**: $E(x_i - y_i) = E(x_i)E(y_i)^{N-1}$
2. C and P:
   for $1 \leq i \leq \ell$ **do**:
   Call $SM(E(x_i - y_i), E(x_i - y_i))$ to compute $E((x_i - y_i)^2)$
3: C:
   Compute $E(|X - Y|^2) = \prod_{i=1}^{\ell} E((x_i - y_i)^2)$

(3) Secure Minimum out of 2 Numbers (SMIN_2) Protocol (**Protocol** 3):

Let $u \in \{0, 1\}^\alpha$ and $v \in \{0, 1\}^\alpha$ be two length-α bit strings, where u_i and v_i $(1 \leq i \leq \alpha)$ denote each bits of u and v, respectively. Therefore, we have $0 \leq u, v \leq 2^\alpha - 1$. Let $[u] = (E(u_1), \cdots, E(u_\alpha)$ and $[v] = (E(v_1), \cdots, E(v_\alpha)$ represent that the encrypted bits

of u and v, where (u_1, u_α) and (v_1, v_α) are the most and least significant bits of u and v, respectively.

Protocol 3: $\mathsf{SMIN}_2([u],[v])) \rightarrow [\mathsf{min}(u,v)]$

Require: C has $[u]$ and $[v]$, where $0 \leq u, v \leq 2^\alpha - 1$; P has sk

1. C:
   - (a) Randomly choose the functionality F
   - (b) **for** $i = 1$ to α **do**: $E(u_i v_i) \leftarrow SM(E(u_i), E(v_i))$
     - **if** F: $u > v$ **then**:
       - $W_i \leftarrow E(u_i)E(u_i v_i)^{N-1}$, $\Gamma_i \leftarrow E(v_i - u_i)E(\hat{r_i}); \hat{r_i} \in Z_N$
     - **else**
       - $W_i \leftarrow E(v_i)E(u_i v_i)^{N-1}$
       - $\Gamma_i \leftarrow E(u_i - v_i)E(\hat{r_i}); \hat{r_i} \in Z_N$
       - $G_i \leftarrow E(u_i \oplus v_i)$, $H_i \leftarrow H_{i-1}^{r_i} G_i; r_i \in_R Z_N$ and $H_0 = E(0)$
       - $\Phi_i \leftarrow E(-1)H_i$, $L_i \leftarrow W_i \Phi_i^{r_i'}; r_i' \in Z_N$
   - (c) $\Gamma^{'} \leftarrow \pi_1(\Gamma)$, $L^{'} \leftarrow \pi_2(L)$
   - (d) Send $\Gamma^{'}$ and $L^{'}$ to P
2. P:
   - (a) $M_i \leftarrow D(L_i')$, for $1 \leq i \leq \alpha$
   - (b) **if** $\exists\, j$ such that $M_j = 1$ **then** $\lambda \leftarrow 1$
     - **else** $\lambda \leftarrow 0$
   - (c) $M_i' \leftarrow \Gamma_i'^{\lambda}$, for $1 \leq i \leq \alpha$
   - (d) Send $M^{'}$ and $E_{pk}(\lambda)$ to C
3. C:
   - (a) $\widetilde{M} \leftarrow \pi_1^{-1}(M^{'})$
   - (b) **for** $i = 1$ to l **do**: $\theta_i \leftarrow \widetilde{M_i} E(\alpha)^{N - \hat{r_i}}$
     - **if** F: $u > v$ **then** $E(min(u,v)_i) \leftarrow E(u_i)\theta_i$
     - **else** $E(\mathsf{min}(u,v)_i) \leftarrow E(v_i)\theta_i$
   - (c) According to $E(\mathsf{min}(u,v)_i)$, C can get $E(\mathsf{min}(u,v))$

(4) Secure Minimum out of k Numbers (SMIN_k) Protocol (**Protocol** 4):

Let $d_i \in \{0,1\}^\alpha (1 \leq i \leq k)$ denote a length-α bit pattern representing a distance, where $d_{i,j} \in \{0,1\}, 1 \leq j \leq \alpha$ denotes a bit of d_i. So, $0 \leq d_i \leq 2^\alpha - 1$. Let $[d_i] = (E(d_{i,1}), \cdots, E(d_{i,\alpha}))(1 \leq i \leq k)$ denote the encrypted vector of the bits in d_i. $d_{i,1}$ and $d_{i,\alpha}$ are the most and least significant bits of d_i. C has k encrypted vectors $([d_1], \cdots, [d_k])$ and P has sk. At the end, no information is revealed to any party.

Protocol 4: $\mathsf{SMIN_k}([d_1], \cdots, [d_k]) \rightarrow [d_{min}]$

Require: C has $([d_1], \cdots, [d_k])$; P has sk

1. C: $[d_i^{'}] \leftarrow [d_i]$, for $1 \leq i \leq k$, $num \leftarrow k$
2. C and P:
   (a) **for** $i = 1$ to $\lceil \log_2 k \rceil$:
   for $1 \leq j \leq \lfloor \frac{num}{2} \rfloor$:
   if $i = 1$
   then: $[d^{'}_{2j-1}] \leftarrow \mathsf{SMIN}_2([d^{'}_{2j-1}], [d^{'}_{2j}])$, $[d^{'}_{2j}] \leftarrow 0$
   else $[d^{'}_{2i(j-1)+1}] \leftarrow \mathsf{SMIN}_2([d^{'}_{2i(j-1)+1}], [d^{'}_{2ij-1}])$, $[d^{'}_{2ij-1}] \leftarrow 0$
   (b) $num \leftarrow \lceil \frac{num}{2} \rceil$
3. C: Set $[d_{min}]$ to $[d_1^{'}]$

2.5 Horizontal Data Partition

We revisit the concept of horizontal data partitioning in the context of two-party computation. Consider two parties, P_1 and P_2, each possessing a dataset, $D_x = \{x_1, x_2, \ldots, x_m\}$ and $D_y = \{y_1, y_2, \ldots, y_n\}$ respectively. Each record $x_i = \{x_{i,1}, x_{i,2}, \ldots, x_{i,\ell}\}$ in D_x and $y_i = \{y_{i,1}, y_{i,2}, \ldots, y_{i,\ell}\}$ in D_y represents an ℓ-dimensional vector, where each dimension corresponds to an attribute value. It is important to note that the two datasets are disjoint.

The datasets are combined to form a joint dataset $D = \{x_1, x_2, \ldots, x_m, y_1, y_2, \ldots, y_n\}$, under the condition that the ℓ attributes in D_x and D_y are identical and follow the same sequence. This unified dataset D is then used for data mining and analysis purposes. The partitioning of D into D_x and D_y exemplifies horizontal data partitioning, characterized by dividing datasets along the rows while retaining the same attribute set across partitions.

2.6 Secret Sharing

Secret-sharing schemes were initially introduced for threshold cases by Blakley and Shamir. In these threshold schemes, the subsets capable of reconstructing the secret are precisely those whose size meets or exceeds a specified minimum number, known as the threshold. The extension of secret-sharing schemes to accommodate general access structures was later developed

and constructed by Ito, Saito, and Nishizeki. These general schemes allow for more complex configurations of subsets to be designated as authorized for secret reconstruction beyond the simple threshold-based criteria.

Secret-sharing schemes are pivotal tools in cryptographic protocols. These schemes are characterized by the presence of a *dealer*, who possesses a secret, a group of n parties, and a specified *access structure*, denoted as A, which is a collection of subsets of parties authorized to access the secret. The objective of a secret-sharing scheme tailored for A is twofold: (1)Enable any subset of parties within A to collaboratively reconstruct the secret using their respective shares. (2)Ensure that any subset not included in A is unable to obtain any information about the secret, thereby preserving its confidentiality.

Initially conceptualized for secure information storage, secret-sharing schemes have since been extensively employed in various domains of cryptography and distributed computing, underscoring their versatility and significance. In this paper, we define two kinds of secret sharing.

- Additive Shares: The notation $[x]$ represents an additive share of a secret x among a group of parties. It signifies that the secret x is decomposed into several parts, such that $x = x^1 + x^2 + \ldots + x^n$, where each x^i is a share held by party P_i. In scenarios where x is shared within a smaller subset of parties, denoted as P_A, the share is specifically represented as $[x]^{P_A}$. This method of sharing ensures that each party only holds a fragment of the secret, and the full secret x can only be reconstructed when all or a sufficient number of these shares are combined.

- Authenticated Shares (SPDZ Shares): The notation $[\![x]\!]$ is used for an authenticated share, often referred to as a SPDZ share. This type of share extends beyond simple additive sharing by including authentication information to verify the integrity and authenticity of

the shares. A SPDZ share comprises a vector of additive shares represented as:

$$[\![x]\!] = ([x], [\alpha], [\alpha \cdot x])$$

Here, $[x]$ is the additive share of the secret, $[\alpha]$ is an additive share of a MAC key used for authentication, and $[\alpha \cdot x])$ is an additive share of the product of the secret and the MAC key. This structure enhances the security of the sharing scheme by enabling the verification of shares without revealing the secret itself.

Secret sharing is a foundational technique in Fluid MPC, enabling secure and distributed computation by dividing sensitive data into multiple shares. Each share reveals no information individually but collectively reconstructs the secret. In Fluid MPC, secret sharing allows computations to be performed collaboratively among parties without exposing their private inputs. This approach ensures both data privacy and robustness against malicious participants, as computations can proceed even if a subset of parties acts adversarially or becomes unavailable. The use of secret sharing also facilitates dynamic resource allocation, improving the efficiency and scalability of the protocol.

2.7 Oblivious Linear Evaluation

Oblivious Linear Evaluation (OLE) is a cryptographic building block that involves two distinct parties: a sender and a receiver. In this primitive, the sender inputs an affine function $f(x) = a + bx$ defined over a finite field $\mathbb{F}$. The receiver, on the other hand, inputs an element $w \in \mathbb{F}$. At the conclusion of the protocol, the receiver learns the value of $f(w)$.

A key feature of OLE is that the sender remains completely unaware of the receiver's input w, and the receiver, in turn, gains no knowledge about the function f beyond the specific value $f(w)$. This characteristic makes OLE a generalization of the well-known Oblivious Transfer

(OT) primitive, expanding its functionality in the domain of linear algebraic operations over finite fields.

Functionality F_{OLE}

Parameters: Finite field $\mathbb{F}_p$, two party P_i with input α and P_j with input x

Extend: On receiving $(Extend, P_i, P_j, \alpha)$ from P_i and $(Extend, P_i, P_j, x)$ from P_j

1. Sample $K \leftarrow \mathbb{F}_p$, Compute $M = \alpha \cdot x - K$.
2. Output K to P_i and M to P_j.

Oblivious Linear Evaluation (OLE) plays a vital role in Fluid MPC by enabling secure and efficient multiplication of private values. In the context of Fluid MPC, OLE allows two parties to collaboratively compute linear operations on their secret-shared inputs without revealing the actual values. This is particularly useful for constructing more complex functionalities, such as matrix multiplications or polynomial evaluations, which are integral to many secure computation tasks. OLE reduces communication overhead and improves the overall efficiency of the protocol, making it well-suited for dynamic and resource-constrained environments.

2.8 Universal Composed Security

2.8.1 The Basic Framework

The Universal Composability (UC) framework defines security by comparing what an adversary can achieve in two different scenarios: a real-world protocol execution and an ideal process. In the ideal process, parties simply submit their inputs to a trusted entity that runs the ideal functionality and then receive their outputs directly from it, with no other interaction taking place.

A protocol is said to be UC-secure if any attack in the real protocol execution (where no

trusted party exists and the parties only interact with each other) does not provide the adversary with more advantages than an attack in this ideal process. In other words, the behavior observed in a real protocol execution should be ”emulatable” in the ideal process.

The term ”emulation” here is defined specifically: it means that for every adversarial strategy in the real protocol, there exists a simulator in the ideal model that can produce a computationally indistinguishable output from what the adversary sees in the real protocol.

This requirement for emulation ensures that a UC-secure protocol maintains its security properties even when composed with other protocols, providing a strong guarantee of security in complex and unpredictable real-world environments. UC security is thus a powerful concept for designing and analyzing cryptographic protocols, ensuring they remain secure under a wide range of conditions and compositions.

In the realm of cryptographic protocols, the standard model of computation includes not only the parties executing the protocol but also an adversary A who controls communication channels and can potentially corrupt parties. A crucial concept in this context is *emulation*.This implies that for every adversary A targeting a real protocol execution, a corresponding *ideal process adversary* or simulator S should exist. The actions of S in the ideal process should yield outputs for the parties that are virtually indistinguishable from those in the real execution.

The Universal Composability (UC) framework builds upon this concept by introducing an extra adversarial element, the *environment* Z. This environment is responsible for generating inputs for all parties, capturing all outputs, and engaging in unlimited interaction with the adversary during the computation. As the name suggests, Z symbolizes the external environment that includes various concurrent protocol executions in addition to the specific protocol in question.

Under the UC framework, a protocol is considered to UC-realize a certain ideal functionality F if, for any *real-life* adversary A involved with the protocol, there exists an *ideal-process adversary* S. This setup should be such that no environment Z can tell apart whether it is in-

teracting with A and the protocol-running parties, or with S and the parties engaging with F in the ideal process. In essence, Z acts as an *interactive distinguisher*, attempting to differentiate between a protocol execution and the ideal process that accesses F. This framework mandates that the *ideal-process adversary* (or simulator) S must interact with Z throughout the computation without the possibility of *rewinding* Z.

Chapter 3

An E-voting System Based on Blockchain

In this chapter, we propose a decentralized e-voting system based on Ethereum. E-voting is an important application of MPC. Specially, we make the following contributions.

- We have designed a decentralized voting protocol capable of resisting malicious activities from voters during certain stages of the voting process. The implementation of this voting scheme is realized through an Ethereum smart contract. This contract utilizes threshold encryption and linkable ring signatures, operating without the need for a trusted third party.
- The protocol ensures maximum privacy, guaranteeing that the results cannot be tallied before the designated end time of the voting period. Additionally, the privacy of each voter's choice is impeccably protected, with the only exception being in the unlikely event of a unanimous conspiracy among all other voters.
- The tallying process is autonomously executed by the smart contract, eliminating the need for traditional election administrators and thereby reducing the risk of human error or manipulation.

3.1 Voting Protocol Description

We present an implementation of the voting protocol over the Ethereum private network with truffle and remix. The election administrator must set up the contract according to the voting rules to the blockchain. Then, publish the codes and provide the contract address. Through this method, all voters can compile the code and verify whether the published code and the smart contract on the Blockchain are the same. The administrator should also publish the list of eligible voters. We assume that each entity has its own Ethereum account to send transactions. In addition, we do not need voters to register their Ethereum accounts, and voters can change their Ethereum account during the voting period. All the data sent to the blockchain must be together with signatures to make sure they are from eligible voters.

3.1.1 Voting Protocol Entities Description

The election system usually involves several entities. For the sake of simplicity, we consider that each entity consists of only one individual, but note that both of them could be thresholded.

- Election administrator: Response for setting up the election; set up the smart contracts; identify the eligible voters with their public keys; publish the list of voters' public keys and the list of candidates.
- Voter: The eligible voters who have a pair of private-secret keys.

The smart contract on blockchain is written in Ethereum's Solidity language. The smart contract has the following functions:

- Control the processing of the election
- Verify if the message is sent by an eligible voter

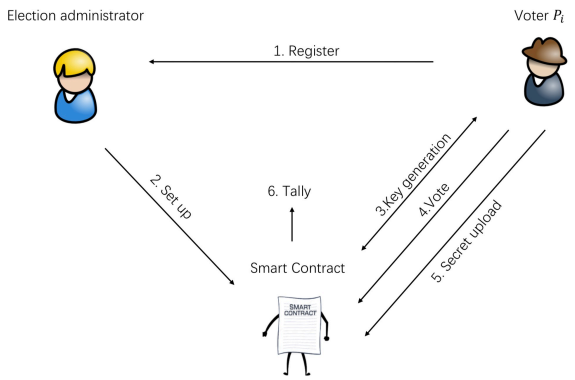

Figure 3.1: Voting stages description

- Store the data of secret share
- Verify the signature (linkable ring signature) of the vote
- Reconstruct the secret key
- Encrypt the vote
- Tally the vote and publish the final result

3.1.2 Voting Protocol Description

There are six stages in our election (as we show in 3.1). The election administrator is the designated owner of the smart contract. The duty of the administrator is to authenticate the voters and set the list of eligible voters with their public keys to the smart contract. The contract allows any users to send messages but only stores the data sent from eligible voters with their signatures and casts votes with correct linkable ring signatures.

In our protocol, we assume that F_q is a finite cyclic group with the order of prime number q. $E(F_q)$ is an elliptic curve over the finite group F_q. G is a base point of the curve $E(F_q)$. l is the order of the base point G. Let H_1 be a cryptographic hash function that can map a number into the finite cycle group F_q. Let H_2 be a cryptographic hash function that can map an input to a point of an elliptic curve [44]. Each stage of the election protocol is described below:

Register: Each voter P_i ($i \in [1, n]$) begins the registration process by generating a pair of public and secret keys (pk_i, sk_i) using a given security parameter. The public key is computed as $pk_i = sk_i G$, where G is a generator of the elliptic curve group or a corresponding parameter in the cryptosystem. Once the key pair is generated, the voter P_i securely transmits their public key pk_i to the election administrator. This ensures that the election administrator can identify registered voters while maintaining the confidentiality of their private keys.

Setup: The election administrator initializes the voting process by performing several critical tasks. First, the administrator sends the complete list of all voters' public keys to the smart contract. This ensures that the contract can verify voters during the election. Additionally, the administrator defines a series of timers to manage the progression of the election process as follows:

- $t_{begingeneration}$: Marks the start of the key generation process. The election administrator sets up the Ethereum contract to initiate this phase at the specified time.
- $t_{finishgeneration}$: Specifies the deadline by which all voters must upload their key generation data.
- $t_{beginvote}$: Signals the Ethereum contract to allow the casting of votes starting from this time.
- $t_{finishvote}$: Establishes the deadline for voters to submit their votes.
- $t_{beginreconstruction}$: Defines the time at which voters can begin uploading their secret shares for threshold decryption.
- $t_{finishreconstruction}$: Sets the final deadline for voters to complete uploading their secret shares.

- $t_{publish}$: Indicates the time by which the Ethereum contract must publish the final election result.

The election administrator also specifies two key parameters for the threshold encryption scheme:

- n: The total number of registered voters.
- t: The minimum number of secret shares required for threshold key reconstruction.

Once these setup processes are complete, the election administrator publicly announces the address of the smart contract and any relevant election information. Additionally, the administrator may set a registration deposit d, which serves as a financial deterrent against malicious behavior. This deposit can be forfeited as a penalty for voters who act dishonestly or fail to comply with the protocol.

Key Generation: Each voter P_i generates their key shares and distributes them securely as follows:

1. *Random Key Selection:* Voter P_i selects a random secret $x_i \in F_q$ uniformly from the field F_q and computes their public key share $g_i = x_i G$, where G is a generator of the elliptic curve group.

2. *Polynomial Creation:* To securely distribute x_i, P_i constructs a random polynomial $f_i(z) \in F_q[z]$ of degree $t-1$ such that $f_i(0) = x_i$. Specifically:

$$f_i(z) = f_{i,0} + f_{i,1}z + \cdots + f_{i,t-1}z^{t-1}, \quad \text{where } f_{i,0} = x_i.$$

3. *Commitments to Coefficients:* Voter P_i computes the commitments $F_{i,j} = f_{i,j}G$ for $j = 0, \ldots, t-1$. Each voter sends $(F_{i,j}, i, j)$ along with a signature generated using their

private key sk_i to the blockchain via the smart contract.

4. *Secret Share Distribution:* Once all $F_{i,j}$ values are uploaded to the blockchain, P_i calculates the secret shares $s'_{i,j} = f_i(j)$ for $j = 1, \ldots, n$, representing the evaluation of $f_i(z)$ at each voter index. These shares are encrypted using the public key pk_j of the recipient voter P_j, resulting in $s_{i,j}$. P_i then sends $(s_{i,j}, i, j)$ with their signature to the blockchain.

5. *Decryption and Publication:* Each voter P_i can decrypt their received shares $s_{j,i}$ for $j \in [1, n]$ and reconstruct their portion of the secret. All commitments $F_{i,j}$ are published on the blockchain for transparency.

6. *Smart Contract Verification:* - The smart contract verifies the signatures of $F_{i,j}$ and $s_{i,j}$ using pk_i. - If any signature fails verification, the smart contract broadcasts an error. - Upon successful verification, all $F_{i,j}$ and $s_{i,j}$ values are published.

7. *Public Key Computation:* The global public key is computed as:

$$g = \sum_{i=1}^{n} F_{i,0},$$

which is publicly available. However, the corresponding secret key $x = \sum_{i=1}^{n} x_i$ remains unknown unless all participants collaborate.

This process ensures secure distribution and verification of the key shares while maintaining the privacy and integrity of the key generation process.

Vote: Each voter P_i computes their vote $V_i^{'}$ based on their choice and the predefined coding rules. The vote $V_i^{'}$ is then encrypted using the global public key g, producing the encrypted vote V_i. Voter P_i sends V_i (the encryption result) along with a linkable ring signature, constructed using the public key list L published in the smart contract on the blockchain.

The smart contract on the blockchain verifies the signatures of all votes to ensure two key

properties: 1. No voter casts more than one ballot (i.e., prevents double voting). 2. All votes originate from eligible voters listed in the public key list.

By leveraging the linkable ring signature, the system ensures voter anonymity while maintaining the integrity of the election process.

Subsecret Generation: Each voter P_i obtains $s'_{j,i}$ by decrypting $s_{i,j}$ using their private key sk_i. Voter P_i verifies the consistency of $s'_{j,i}$ for $j \in [1, n]$ with the published values on the smart contract by checking the following equation:

$$s'_{j,i} \cdot G = \sum_{l=0}^{t-1} F_{j,l} \cdot i^l,$$

where G is the generator of the elliptic curve group and $F_{j,l}$ are the commitments to the coefficients of the polynomial provided by P_j.

After successful verification, P_i computes their share of the secret x, denoted as s_i, by summing up all verified $s'_{j,i}$ for $j \in [1, n]$:

$$s_i = \sum_{j=1}^{n} s'_{j,i}.$$

This process ensures that each voter correctly reconstructs their individual share of the global secret x while maintaining consistency and correctness of the distributed secrets.

Secret Upload: Each voter P_i uploads their share s_i along with their identifier i to the smart contract, signing the submission using their private key sk_i.

The smart contract verifies the signatures to ensure the authenticity of the submitted shares. Once t valid shares s_i are received, the smart contract reconstructs the global secret key x using

the threshold encryption system. The reconstruction is performed as follows:

$$x = \sum_{j=1}^{t} s_j \cdot \prod_{h=1, h \neq j}^{t} \frac{h}{h-j} \pmod{l},$$

where l is the modulus, and s_j are the received shares.

This process ensures that the secret x can be reconstructed securely using the threshold t, without requiring all n shares, thus preserving the robustness and fault tolerance of the system.

Tally: The smart contract uses the reconstructed secret key x to decrypt all encrypted votes, obtaining the real votes while preserving their anonymity. After decrypting, the smart contract tallies the votes to calculate the final result R. The final result is then published on the blockchain, ensuring transparency and verifiability.

3.2 Voting Protocol Analysis

3.2.1 Correctness and Security Analysis

We will discuss the protocol in the following aspects as mentioned in [36]:

- **Correctness**: The smart contracts deployed on the Ethereum network provide a decentralized and tamper-proof computing environment, ensuring that the final result is computed correctly without the possibility of alteration.
- **Robustness**: In our protocol, semi-honest voters cannot disrupt the voting process. Even if some malicious voters fail to upload their secret shares during the secret upload phase, the final result will still be computed correctly due to the threshold encryption scheme. Transactions with incorrect signatures or invalid linkable ring signatures are automatically rejected by the smart contract, further ensuring the protocol's robustness.

- **Privacy**: Each vote sent to the blockchain is accompanied by a signature generated using a private key, with the corresponding public key already registered. The use of linkable ring signatures ensures that no entity, including voters, candidates, or the election administrator, can identify the origin of a signature with a probability greater than $1/n$, where n is the total number of voters. This guarantees strong voter anonymity.

- **Double-voting Avoidance**: The linkable ring signature mechanism allows the smart contract to detect and prevent double voting. Legal signature features are stored on the blockchain, and any new signature is verified against the stored features to ensure it has not been used before. As a result, each eligible voter can cast their vote only once.

- **Validity**: All eligible voters must register their public keys with the election administrator and securely keep their private keys. Messages sent to the smart contract must be signed with the private keys, and no valid signature can be generated without knowledge of the corresponding private key. This ensures that all submitted ballots are valid.

- **Fairness**: Votes stored on the public ledger remain encrypted and are only decrypted during the tally phase by the smart contract. This ensures that no one, including voters and administrators, can access the results before the voting process is complete. Intermediate results cannot be inferred, preventing any undue influence on the ongoing voting process.

- **Verifiability**: Anyone with access to the address of the voting smart contract can verify that all ballots are counted correctly. Additionally, voters can check if their votes have been successfully cast by verifying the presence of their ballots in the smart contract.

In order to show the security of our protocol, we will discuss two typical attacks.

- **Man-in-middle Attacks:** All messages sent by voters are signed by private key. The correctness of messages is guaranteed by the signature algorithm. In this way, all messages could not be forged or tampered with. In addition, all public keys are recorded

on the blockchain so that attackers cannot replace the public key to achieve the goal of attacking.

- **Dos Attacks:** In our system, the attacker could not have the ability to destroy all nodes of the blockchain network. So, our protocol could resist the Dos attack. If the network service is provided in a relatively centralized manner, a DoS attack is feasible. In addition, the server's ability to handle large numbers of requests is relatively limited. Distributing services on different nodes is one of the solutions for DoS attacks because it is almost impossible for an attacker to destroy all servers. The underlying framework of the blockchain adopted by this solution can ensure that the system resists DOS attacks.

3.2.2 Decentralized and Trustless Analysis

The voting system proposed in this paper is a decentralized voting system, mainly reflected in two ways:

(1) The voting program in this paper is set up as a smart contract, and the smart contract is carried on the blockchain network. The blockchain network interacts with a peer-to-peer network. Therefore, the system proposed in this paper is a decentralized and trustless voting system. This is why all programs deployed on the blockchain become distributed applications (Dapps).

(2) In this voting protocol, there is no central role. The central role is that of the person who plays a vital role in the election process. If the role maliciously destroys the voting, the entire voting result will be tampered with. For example, in some election agreements, the election administrator performs crucial operations in the voting process□such as decrypting the voting result. However, there is no such role in the protocol of this paper, and the trust in the protocol is dispersed to all voters. Therefore, from this perspective, the agreement on this topic is a decentralized and trustless voting system.

3.2.3 Time Cost Analysis

The implementation of each function in the protocol is written in Python. To evaluate the computational performance, we tested the program on a MacBook Pro running macOS 10.13.6. The machine is equipped with an Intel Core i5 processor (4 cores, 2.9 GHz) and 8GB DDR3 RAM. All measurements are recorded in milliseconds for accuracy.

To analyze the scalability of the protocol, we conducted tests with varying numbers of voters, setting $n = 10, 20, 30, 40$. Here, n represents the total number of voters, and $t = 0.7n$ indicates the threshold of voters required to correctly upload their secrets. This ensures that even with some malicious or offline voters, the protocol remains robust if t voters participate honestly.

For $n = 30$, Table III provides the average computation time for each operation per voter. The operations include key generation, polynomial computation, signature generation, verification of other voters' shares, subsecret reconstruction, and ballot creation.

Table 3.1: Average running time for each operation ($n = 30$)

Operation Description	Time (ms)
A: Generate public/private key pair	36.19
B: Compute $F_{i,j}$, $f_i(j)$ with signature	4,900.23
C: Verify $f_i(j)$ from other voters	0.23
D: Compute global public key	4.01
E: Reconstruct subsecret	0.08
F: Create ballot with signature	558.08

To investigate how the computation time scales with the number of voters, we measured the running time for different operations at $n = 10, 20, 30, 40$. These results are visualized in Figure 3, where the x-axis represents the number of voters, and the y-axis shows the computation time

(in milliseconds) on a logarithmic scale.

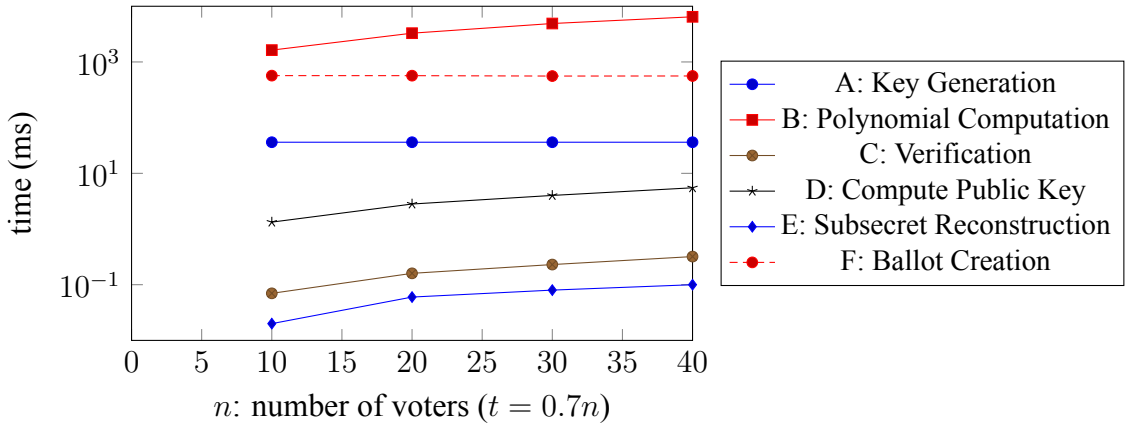

Figure 3.2: Running time of different operations across voter counts

From Figure 3, it is evident that the computation times for most operations, such as key generation (A) and ballot creation (F), remain constant as the number of voters increases. This indicates these operations are independent of n. In contrast, operations such as polynomial computation (B) and global public key computation (D) scale linearly with n, as they involve data from all voters. Verification (C) and subsecret reconstruction (E) are computationally lightweight, even with increasing n, highlighting their efficiency.

Overall, the execution times for all operations remain within acceptable levels, ranging from milliseconds to a few seconds. This ensures the protocol is practical and efficient, even for moderately sized elections, and provides a seamless voting experience for all participants.

3.3 Voting Protocol Comparison

In this section, we compare our protocol with other protocols based on smart contracts.

In protocol II (McCorry [66]), if any voter behaves maliciously during the voting process, the entire election can be disrupted, and the votes cannot be tallied correctly. In contrast, protocol I (our protocol) is robust against malicious voters. Any malicious behavior can be detected,

Table 3.2: Protocol comparison

Protocol	Handles Malicious Voters	Self-Tally	Requires Whitelist	Rounds
I: Our protocol	Yes	Yes	No	3
II: McCorry [66]	No (Destroyed)	Yes	Yes	2
III: Yu [90]	Yes	No	Yes	1

and such voters can be excluded from the protocol. Moreover, during the key generation phase, even if some voters act maliciously, as long as there are enough honest voters (exceeding the threshold set by the election administrator) who upload their subsecrets, the protocol can still produce a correct final result.

Another distinction lies in the tallying process. In protocol I, the tallying is performed autonomously by the smart contract, ensuring decentralization and removing reliance on a single party. In protocol III (Yu [90]), however, the tallying process is handled by an election administrator. If the administrator fails to act, the entire voting process will be compromised.

Protocol I also eliminates the need for a whitelist containing all eligible Ethereum addresses. Instead, the protocol uses signatures to verify voter identities, allowing voters to change their Ethereum addresses during the voting process, which enhances voter privacy. This functionality is a key advantage over protocols like II and III, which rely on fixed whitelists.

However, protocol I involves three communication rounds, making it slightly more complex than the other protocols. Despite this, it provides stronger privacy protections compared to the alternatives.

Chapter 4

Two-party k-means Clustering Protocol

In this chapter, we present an efficient two-party privacy-preserving collaborative k-means clustering protocol with the following properties.

- Each party's database is stored in its encrypted form in the cloud.
- The k-means clustering protocol needs to work on the combined set of records of both parties (i.e., the overall dataset is horizontally partitioned to the two parties).
- The encrypted clustering result is sent to each party for decryption so as to keep it private from the cloud or any other party.

To achieve the above properties, the underlying encryption algorithm has to support some specific operations, including distance computation, distance comparison, and centroids recomputation on encrypted data. In this paper, we use Paillier encryption as the underlying encryption algorithm and extend it to support various operations.

4.1 Protocol Description

The high-level idea is as follows. For distance computation, we consider Euclidean distance. We require that both addition and multiplication operations are performed on ciphertext, and the result is also encrypted. However, Paillier encryption only supports additive homomorphic operation, i.e. $E(x)E(y) = E(x + y)$. We adopt the Secure Multiplication (SM) protocol introduced in [34] to output $E(xy)$, with one-round interaction with the corresponding secret key owner, given input of $E(x)$ and $E(y)$. Then $E((x - y)^2)$ can be computed by running SM with input $E(x - y)$ and $E(x - y)$, where $E(x - y) = E(x)E(y)^{N-1}$.

For distance comparison, it requires order-preserving encryption. Paillier encryption is obviously not appropriate. The idea is to compare two distance values bit by bit, from the most significant bit to the least one in encrypted form.

For centroid re-computation, we can deduce it to a protocol with input x_1 by P_1, x_2 by P_2, x_3^* by C to output $\frac{x_1+x_2}{x_3^*}$ to both P_1 and P_2 but not C. According to the protocol introduced in [75] with the same input as the proposed protocol's, output $\frac{x_1+x_2}{x_3^*}$ to P_1, P_2 and C. We then set up a garbled circuit to compute $\frac{x_1+x_2}{x_3 \oplus x_4}$ and regard SHA256 as the commit method to meet the requirement.

In addition to the functionalities discussed above, efficiency is another important aspect. First of all, the encryption algorithm used for each data owner should not be complicated. The corresponding encrypted data size should be as small as possible. Then, to cluster the encrypted data, both the total number of interactions among P_1, P_2 and C, and the computation executed on P_1, P_2 and C should be as little as possible. Lastly, the communication payload (the data transferred among P_1, P_2 and C) should be low, although this may not be as important as other concerns.

In the design, each party executes Paillier encryption once. The total number of interactions

is bounded by $O((m+n)k)$, where m and n are the total numbers of records provided by P_1 and P_2, respectively, for each iteration in k-means clustering. In the step of k centroids re-computation, we choose to use the garbled circuit to achieve the computation of $\frac{x_1+x_2}{x_3^*}$. Once the corresponding garbled circuit is built, the computation is very fast.

4.1.1 Framework and Notation

The framework is illustrated in Figure 4.1. P_1 has secret key sk_1 and P_2 has secret key sk_2. Each has a dataset, D_x and D_y, respectively. D_x has m data points and D_y has n data points. Every data point is a ℓ-dimensional vector. In other words, every data point has ℓ attributes. (1) To outsource the storage while guaranteeing privacy, D_x and D_y are encrypted by pk_1 and pk_2, and uploaded to C by P_1 and P_2, respectively. (2) C randomly chooses k centroids M for k clusters. (3) C computes distances between each centroid and each record in D_x and D_y, with the help of P_1 and P_2. (4) By distance comparison, all records can be clustered to the nearest centroid. (5) C, P_1 and P_2 jointly re-compute the new set of k centroids. Note that the k centroids are known only to P_1 and P_2. Once the distances of the new k centroids to the previous k centroids are all within a threshold value vector $\tau = \{\tau_c | 1 \leq c \leq k\}$, P_1 and P_2 will request C for the clustering records for decryption. Then the protocol ends. Otherwise, P_1 and P_2 encrypt the new k centroids by their public keys and upload the ciphertext to C, respectively. Then P_1 and P_2 ask C to compute distances again.

We allow the dataset D to be horizontal partitioned between P_1 and P_2, each of which has D_x and D_y. If not explicitly specified, the Euclidean distance is used in our k-means clustering algorithm. Only numerical data is supported. The threshold value vector to end our protocol τ should be fine-tuned according to the applications. Paillier Encryption is used to encrypt data. (pk_1, sk_1) and (pk_2, sk_2) are P_1 and P_2's public and secret key pairs generated by K.

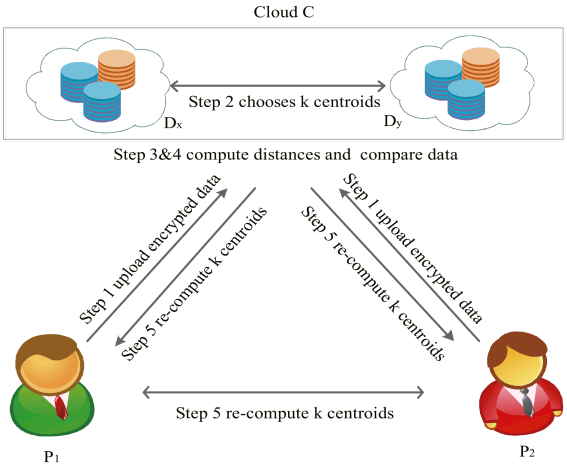

Figure 4.1: Framework of privacy-preserving collaborative k-means clustering protocol

4.1.2 Two-party k-means Collaborative Clustering Protocol

k-means clustering algorithm is a classical clustering algorithm based on distance. To support clustering under our setting, we extend it to collaborative clustering here. Denote the training data of the two parties, P_1 and P_2, by $\{x_i \in N^\ell | 1 \leq i \leq \ell\}$ and $\{y_i \in N^\ell | 1 \leq i \leq \ell\}$. The algorithm is illustrated in Protocol 5. We shall focus on how to execute the computations, such as multiplication, addition, comparison, etc., in ciphertext mode.

Protocol 5: Two-party k-means collaborative clustering algorithm

1. P_1 and P_2 share k centroids $M = \{\mu_c \in N^\ell | 1 \leq c \leq k\}$ randomly selected.
2. Repeat the following algorithm to converge {
   For each $i \in \{1, \cdots, \ell\}$, compute $\text{argmin}_c ||x_i - \mu_c||^2, 1 \leq c \leq k$,
   $C_c^x = C_c^x \bigcup \{x_i\}$ which records all x_i's that has the nearest distance to μ_c.
   Similarly compute $\text{argmin}_c ||y_i - \mu_c||^2, 1 \leq c \leq k$, and $C_c^y = C_c^y \bigcup \{y_i\}$.
   For every cluster c, recompute the new centroid $\mu_c := \frac{sum_x + sum_y}{|C_c^x| + |C_c^y|}$, where
   $sum_x = \sum_{x_i \in C_c^x} x_i$, $sum_y = \sum_{y_i \in C_c^y} y_i$,
   $|C_c^x|$ and $|C_c^y|$ denote the numbers of x_i's and y_i's in C_c^x and C_c^y. }

* $|| \cdot ||$ denotes the Euclidean distance.
* The condition to converge is that C_c^x and C_c^y keep fixed.

4.1.3 Secure Garbled Circuit Protocol Supporting $\frac{x_1+x_2}{x_3^*}$

We symbolize the three parties in the protocol by P_1, P_2 and C, their respective inputs by x_1, x_2 or x_3^* and their collective output by y. They collaborate to compute the following function securely, $y = f(x_1, x_2, x_3^*) = \frac{x_1+x_2}{x_3^*}$. To simplify the problem, we assume that $|x_i| = |y| = m$. In the following, we target the following: P_1 and P_2 can learn the same output y while C cannot get the output y with these garbled values. This protocol uses a scheme of garbling, a four-tuple algorithm $\delta = (\mathsf{Gb}, \mathsf{En}, \mathsf{De}, \mathsf{Ev})$, as the underlying algorithm. Gb is a randomized garbling algorithm that performs the transformation. En and De are encoding and decoding algorithms, respectively. Ev is the algorithm that derives garbled output on the basis of garbled input and garbled circuit.

We firstly transform *division* to *multiplication*. We use the SHA256 hash function as the commit method. f' is the function of $\frac{a+b}{c \oplus d}$ which a, b, c and d are all 32 bits and the result is 65 bits. We also set up a garble circuit F by using AND/NOT/OR/XOR gates whose total number is 12,470. The details of the protocol are described in Protocol 6.

4.1.4 Details of the Privacy-preserving Collaborative k-means Clustering Protocol

In this section, we present the detailed steps of the proposed privacy-preserving collaborative k-means clustering protocol.

Step 1 P_1 and P_2 upload encrypted data

P_1 and P_2 encrypt their data D_x and D_y to C_x and C_y, and upload to the cloud C, respectively.

$C_x = \{C_{x_i} | 1 \leq i \leq m\}$, where $C_{x_i} = \{C_{x_{ij}} = E_{pk_1}(x_{ij}) | 1 \leq j \leq \ell\}$

$C_y = \{C_{y_i} | 1 \leq i \leq n\}$, where $C_{y_i} = \{C_{y_{ij}} = E_{pk_2}(y_{ij}) | 1 \leq j \leq \ell\}$

Protocol 6: $\mathsf{SC}(x_1, x_2, x_3^*) \rightarrow y$

Require: In our experiment, we calculate $\frac{x_1+x_2}{x_3^*}$ which P_1 has x_1, P_2 has x_2 and C has x_3^*. We use SH256 as the commit method. $f^{'}$ is the function of $\frac{a+b}{c \oplus d}$ which a, b, c and d are all of 32 bits and the result is of 128 bits.

1. C:

   (a) Sampling a common random string, can also be expressed as crs for the commitment scheme and randomly secret-shares his input x_3^* as $x_3^* = x_3 \oplus x_4$.

   (b) Send x_3 to P_1 and x_4 to P_2 and broadcast common random string b to both parties.

2. P_1:

   Choose random pseudo-random function seed $r \leftarrow \{0,1\}^k$ and send it to P_2.

3. P_1 and P_2:

   (a) Garble the function $f^{'}$ via $\mathsf{Gb}^1(1^\lambda, f^{'}) \rightarrow (F, e, d)$ where F is the garble circuit, e is the encoding array and d is the decoding array.

   (b) For $j \in [0, 128]$, $a \in \{0,1\}$. generate the following commitments:

$$\sigma_j^a = e[j, b[j] \oplus a], C_j^a = hash(\sigma_j^a)$$

   (c) Both P_1 and P_2 send the following values to C:

$$(b[65...128], F, \{C_j^a\}_{j,a})$$

4. C:

   Abort if P_1 and P_2 report different values for these items.

5. P_1 and P_2:

   (a) P_1 sends de-commitment $\sigma_j^{x_1[j] \oplus b[j]}$ and $\sigma_{2m+j}^{x_3[j] \oplus b[2m+j]}$ to C

   (b) P_2 sends de-commitment $\sigma_{m+j}^{x_2[j] \oplus b[m+j]}$ and $\sigma_{3m+j}^{x_4[j] \oplus b[3m+j]}$ to C

6. C:

   (a) For $j \in [128]$, compute $C_j^{a'} = hash(\sigma_j^a)$, check $C_j^{a'} = C_j^a$, for the appropriate $o[j]$. If not, then abort. Similarly, C knows the values $b[2m+1, \cdots, 4m]$, and aborts if P_1 or P_2 did not open the "expected" commitments $\sigma_{2m+j}^{x1[j] \oplus b[2m+j]}$ and $\sigma_{3m+j}^{x1[j] \oplus b[3m+j]}$ corresponding to the garbled encodings of x_3 and x_4

   (b) Run $Y \longleftarrow \mathsf{Ev}(F, X)$ and broadcast Y to P_1 and P_2

7. P_1 and P_2:

   Compute $y = \mathsf{De}(d, Y)$. If $y \neq \perp$, then output y. Otherwise, abort

Step 2 Cloud C randomly chooses k centroids for k clusters

C randomly chooses the set of k centroids $\Phi = \{\mu_c | 1 \leq c \leq k\}$, where each $\mu_c = \{u_{cj} | 1 \leq j \leq \ell\}$. Encrypt it using P_1 and $P_2's$ public keys, pk_1 and pk_2, respectively, and store as C^1_μ and C^2_μ.

$C^1_\mu = \{C^1_{\mu_c} | 1 \leq c \leq k\}$, where $C^1_{\mu_c} = \{C^1_{\mu_{cj}} = E_{pk_1}(\mu_{cj}) | 1 \leq j \leq \ell\}$

$C^2_\mu = \{C^2_{\mu_c} | 1 \leq c \leq k\}$, where $C^2_{\mu_c} = \{C^2_{\mu_{cj}} = E_{pk_2}(\mu_{cj}) | 1 \leq j \leq \ell\}$

C^1_μ and C^2_μ are sent to P_1 and P_2, respectively. After decryption, Φ is stored by P_1 and P_2, respectively, for comparison use later in Step 5.

Step 3 Cloud C computes distances

C computes all encrypted distances between each record C_{x_i} and each centroid $C^1_{\mu_c}$, and distances between each record C_{y_i} and $C^2_{\mu_c}$, as follows.

$$CD^1 = \{CD^1_i = \{cd^1_{ic} = \mathsf{SSED}(C_{x_i}, C^1_{\mu_c}) | 1 \leq c \leq k\} | 1 \leq i \leq m\}$$

$$CD^2 = \{CD^2_i = \{cd^2_{ic} = \mathsf{SSED}(C_{x_i}, C^2_{\mu_c}) | 1 \leq c \leq k\} | 1 \leq i \leq m\}$$

Specifically, C and P_1 run SSED to compute the distance between each x_i and μ_c in encrypted form, denoted by cd^1_{id}. Similarly, C and P_2 run SSED to compute the distance between each y_i and μ_c in encrypted form, denoted by cd^2_{ic}. All distances from x_i to μ_c are stored in CD^1_i, and those from y_i to μ_c are stored in CD^2_i.

Step 4 Cloud C clusters records to k clusters for P_1 and P_2

By comparing the distances in CD^1_i and CD^2_i, x_i and y_i will be clustered to the cth cluster if and only if cd^1_{ic} and cd^2_{ic} are the smallest distance in CD^1_i and CD^2_i, respectively. For encrypted distance comparison, C runs $\mathsf{SMIN}_k(CD^1_i)$ with P_1 and $\mathsf{SMIN}_k(CD^2_i)$ with P_2, as follows. Then, C_{x_i} and C_{y_i} will be assigned to CL^1_c and CL^2_c, respectively. As a result, each CL^1_c stores the encrypted data C_{x_i} whose distance to the cth centroid μ_c is

the shortest among all the k centroids. In other words, x_i belongs to the cth cluster. The same as CL_c^2.

$$CL_1 = \{CL_c^1 = \{C_{x_i} | cd_{ic}^1 = min(CD_i^1) = \mathsf{SMIN}_k(CD_i^1)\} | 1 \leq c \leq k\}$$

$$CL_2 = \{CL_c^2 = \{C_{x_i} | cd_{ic}^2 = min(CD_i^2) = \mathsf{SMIN}_k(CD_i^2)\} | 1 \leq c \leq k\}$$

Step 5 Cloud C, P_1 and P_2 jointly re-compute k centroids

Now, C is required to find the new centroid within each cluster, given all the data in the cluster. Note that there are two sub-clusters in each cluster CL_c^1 and CL_c^2 as the data in those two sub-clusters are encrypted by different public keys pk_1 and pk_2. Therefore, the computation of $\mu_{cj}^{'} = \frac{\sum_{i,\text{s.t.},C_{x_i} \in CL_c^1} x_{ij} + \sum_{i,\text{s.t.},C_{y_i} \in CL_c^2} y_{ij}}{|CL_c^1| + |CL_c^2|}$ in not straightforward. Our idea is to send CL_c^1 and CL_c^2 to P_1 and P_2 for decryption first. Let L_c^1 and L_c^2 denote the decrypted data in the cth cluster owned by P_1 and P_2, respectively. Then we have

$$L_c^1 = \{x_i = \{x_{ij} = \mathsf{D}_{sk_1}(C_{x_{ij}}) | 1 \leq j \leq \ell\} | C_{x_i} \in CL_c^1\}$$

$$L_c^2 = \{y_i = \{y_{ij} = \mathsf{D}_{sk_2}(C_{y_{ij}}) | 1 \leq j \leq \ell\} | C_{y_i} \in CL_c^2\}$$

Then, P_1, P_2 and C jointly run $\mathsf{SC}(\sum_{i,\text{s.t.},C_{x_i} \in CL_c^1} x_{ij}, \sum_{i,\text{s.t.},C_{y_i} \in CL_c^2} y_{ij}$, $A|L_c^1| + |L_c^1|)$ to calculate each component of the c-th centroid $\mu_{cj}^{'}$. SC guarantees both P_1 and P_2 can get all the new k centroids in plaintext. Let $\Phi^{'} = \{\mu_c^{'} | 1 \leq c \leq k\}$, where $\mu_c^{'} = \{\mu_{cj}^{'} | 1 \leq j \leq \ell\}$. Denote $\Phi - \Phi^{'} = \{|\mu_c - \mu_c^{'}| | 1 \leq c \leq k\}$ the distance set of the newly generated k centroids to the previous k centroids, where $|\mu_c - \mu_c^{'}| = \sum_{j=1}^{i}(|\mu_{cj} - \mu_{cj}^{'}|)$.

Step 6 P_1 and P_2 decrypt CL_1 and CL_2 or go to Step 3.

Once $|\mu_c - \mu_c^{'}| \leq \tau_c$ for each c, P_1 and P_2 request C for the clustered records CL_1 and CL_2 for decryption, respectively. Then, the protocol ends. Otherwise, P_1 and P_2 encrypt

the new k centroids by their public keys and upload them to C. Then go to Step 3 and iterate.

4.2 Protocol Security Analysis

4.2.1 Security Model

During the first 4 steps described in Section 3.4, P_1 and P_2 interact with C, respectively, with no interactions between P_1 and P_2 or among P_1, P_2, and C. Note that P_1 and P_2 outsource the encrypted distance computation and comparison to C. Since traditional Paillier encryption cannot support the above two operations at the same time, help from P_1 and P_2 is required, which introduces the extra interactions between P_1 and C, P_2 and C. Therefore, the security underlying is essentially secure computation outsourcing. In a semi-honest model, the honest-but-curious cloud will honestly execute the outsourced computation protocols while being motivated to learn any information of P_1 and P_2's raw data or the computation result for financial gains.

In the last step, where $F(x_1, x_2, x_3^*) = \frac{x_1+x_2}{x_3^*}$ is required with each input x_1, x_2, x_3^* of P_1, P_2 and C, it is indeed a three-party secure computation. We adapt the model of 1-out-of-3 active security where C is actively corrupted [67].

4.2.2 Security Analysis

As for Paillier encryption, we cannot decrypt the ciphertext without the private key. So, each date owner encrypts the data they own. Both the cloud and any other party cannot decrypt it. Due to the semantic security of the Paillier cryptosystem, one party's input is protected from the other party.

Here, the security of the scheme under the semi-honest model is verified mainly by attack

mode, and in the secure circuit protocol, even if either party is a malicious party, the scheme is still safe. There are two main types of attacks: two-party attacks and attacks from the cloud platform.

P_1 and P_2's data are encrypted with their own public keys and uploaded to the cloud server. Even if they get the other party's ciphertext, they still cannot get the plaintext through the ciphertext. Cloud platforms may have some background knowledge about the data, so a statistical attack may be feasible. First, because Paillier is a non-deterministic encryption, even if it is the same plaintext, the encrypted ciphertext is different. During the implementation of the protocol, the data obtained by the cloud platform is in the form of a ciphertext, so the cloud cannot obtain any information in the plaintext through the ciphertext.

We can prove the SC protocol is secure against one single malicious party as follows:

Assume that P_1 is corrupted (the case for P_2 is similar). The other two parties are honest. We need to prove that all environments cannot be distinguished whether the protocol is executed under actual conditions or in an ideal situation. The information available to the environment consists mainly of two parts: the information sent by the malicious party and the final output of the protocol based on the information it obtains. As long as the information obtained by the environment cannot be used to distinguish the two conditions (actual and ideal), the environment cannot be distinguished whether the protocol is executed under an actual condition or an ideal condition.

The simulator takes the role as honest P_2 and C obtaining their inputs x_2 and x_{3^*} on their behalf. Then the simulator sends a random value r_{crs} and a random share r_{x_3} to P_1; it can abort if P_1 has changed the commitment; otherwise it extracts $x_1 = o \oplus b[1...m]$ and sends it to the ideal functionality F_f. It receives y, and sends Y to P_1. We can get the $View^{env}_{real} = \{crs, x_3, Y, y\}$ and $View^{env}_{ideal} = \{r_{crs}, r_{x_3}, Y, y\}$. Because crs and x_3 are pseudorandom numbers and r_{crs} and r_{x_3} are random numbers, all environments cannot distinguish them with non-negligible probability.

Next, we consider a corrupted C: The simulator takes the role as both honest P_1 and P_2. It extracts $x_3^* = x_3 \oplus x_4$ and sends it to F_f, obtaining the output y in return. Then it produces a simulated garbled circuit/input(F, X) using y. We can get the $View_{real}^{env} = \{C_j^{o[j]}, o, y\}$ and $View_{ideal}^{env} = \{C_j^{r_o[j]}, r_o, y\}$. Because o are pseudorandom numbers and r_{crs} and r_o are random numbers, all environments cannot distinguish them with non-negligible probability.

Therefore, the SC protocol is secure against a single malicious party.

4.3 Protocol Performance Analysis

The time consumption of the k-means clustering algorithm with privacy protection is mainly divided into three parts: time consumption of the client, communication consumption, and time consumption of the server, where the client and server time consumption include the time consumption of the initialization phase and the protocol running phase. Because this paper is different from the method used in [65], it can only be compared from a macro perspective. The comparison mainly includes two aspects: one is theoretical complexity analysis, including time complexity, space complexity, and communication complexity, and the other is the comparison of test results in experiments. The number of different iterations will affect the overall performance of the experiment, so one iteration will be considered.

4.3.1 Theoretical Analysis

In the paper, we assume that cloud C has extensive computational power. Thus, the computational time used by C is not considered. Each data owner does not need to store the ciphertext; they just encrypt the message with the public key and decrypt the ciphertext with the private key.

Every iteration, data owners will provide some information and this information will be

Table 4.1: Time complexity comparison with [65]

	this paper	[65]
Encryption	$O(n*l)$	$O(n*l)$
Euclidean distance	$O(n*k*l)$	$O(n*k*l)$
Minimum distance	$O(n*k*\alpha)$	$O(n*k)$
$\frac{x_1+x_2}{x_3^*}$	$O(4m*k*l)$	$O(n)$

* Euclidean distance and minimum distance correspond to SSED and $\mathsf{SMIN_k}$, respectively in this paper.

Table 4.2: Space complexity comparison with [65]

	this paper	[65]
Encryption	$O(n*l)$	$O(l*n)$
Euclidean distance	$O(n*k*l)$	$O(n*k*l)$
Minimum distance	$O(n*k)$	$O(n*k)$
$\frac{x_1+x_2}{x_3^*}$	$O(k*l)$	$O(k*l)$

computed in each iteration, and P_1, P_2 and C will recalculate the cluster. We assume that t is the times of iteration, n is the data size of P_1 and P_2, l is the dimension of the data, α is the binary bits of the data, m is the bits of the garbled circuit.

In each iteration, firstly, each data owner will execute SEED protocol and $\mathsf{SMIN_k}$ protocol with C. There are two interactions in SEED protocol and two interactions in $\mathsf{SMIN_k}$ protocol. Then, P_1, P_2 and C will execute 6 times interactions in SC protocol. Finally, each data owner will execute 1 times interactions when they upload new centroids to C. The time complexity comparison with [65] is shown in Table 4.1.

When P_1 and P_2 both upload the encrypted data to the cloud, two parties can delete the data without storing the plaintext data or ciphertext data. They only need to store their public and private keys. Cloud C needs to store all ciphertexts. For each data point, center point distance, and round of iterative categorization, the cloud C needs the required storage space to record the clustering results. The space complexity comparison with [65] is shown in Table 4.2.

Table 4.3: Communication complexity comparison with [65]

	this paper	[65]
Encryption	$O(1)$	$O(1)$
Euclidean distance	$O(n * l * k)$	$O(1)$
Minimum distance	$O(n * k)$	$O(k + n)$
$\frac{x_1+x_2}{x_3^*}$	$O(k * l)$	$O(1)$

In the first step and the second step, P_1 and P_2 upload their own data and the encrypted cluster center to the cloud, respectively, which requires four iterations. The following is an analysis of the communication complexity for each iteration. In the third step, P_1 and P_2 are respectively executed in the secure distance calculation protocol with the cloud. In the fourth step, P_1 and P_2 need to interact with each other when executing security comparison protocol with the cloud. In the fifth step, P_1, P_2 and cloud execute a secure circuit protocol that recalculates the clustering center. The communication complexity comparison with [65] is shown in Table 4.3.

4.3.2 Experimental Analysis

The framework used by the k-means clustering algorithm with privacy protection proposed in this paper was first proposed in the [65]. Compared with the clustering algorithms in other frameworks, the clustering algorithm under the same framework can be easily compared. Therefore, we mainly compare the protocol with the [65]. In order to ensure the reliability of the experimental comparison, both schemes were run in the same experimental environment. The evaluation criteria of the two schemes will be introduced below, and a comparative analysis of the experimental results will be carried out.

Table 4.4: Time of encryption of the proposed protocol

data size	3-dimension(ms)	7-dimension(ms)
500	1,730	4,227
1,000	3,603	8,330
2,000	7,504	16,287
5,000	17,690	35,917
10,000	34,929	80,543

Table 4.5: Time of encryption of paper [65]

data size	3-dimension(ms)	7-dimension(ms)
500	2,391	5,084
1,000	4,587	11,468
2,000	9,413	21,487
5,000	20,657	43,186
10,000	40,894	87,461

4.3.3 Analysis of Results

In theory, the performance of the protocol is better than those in the literature in terms of time complexity, space complexity and communication complexity [65]. Now, we want to verify the results based on experiments. We first compare the encryption time consumption of the two schemes. In the two encryption methods used in [65], all plaintext data must be encrypted once by the improved Liu encryption scheme and once by Paillier encryption scheme. All the plaintext data in this paper's scheme only needs one Paillier encryption. In theory, the encryption time in the scheme in this paper should be faster than the literature [65]. And because Paillier's operation is on the group, there are many exponential operations, and the improved Liu encryption scheme is linear, so most of the encryption time is consumed by Paillier encryption. Therefore, the encryption time consumption in this paper will be slightly less than the encryption time consumption in [65], but there is no order of magnitude difference in time. The experimental results provide strong support for this conclusion. The encryption time consumption in [65] is shown in Table 4.5. The encryption time consumption of this paper is shown in Table 4.4.

Next, we counted and compared the time spent in an iteration. In theory, the cloud platform

Table 4.6: Time comparison with [65] in one iteration

data size	this paper(ms)	[65](ms)
500	23,872	13,279
1,000	25,095	20,528
2,000	25,572	27,276
5,000	32,640	33,508
10,000	42,746	51,324

introduced in this paper has improved the powerful computing power and should be slightly better than the operating efficiency in the literature [65]. Because the cloud platform is composed of 30 PCs and one server, it is necessary to perform task division, task scheduling, and data recovery for each machine during the processing of tasks. These operations also consume part of the time. When there are more data points, the time of one iteration will be longer, and the proportion of time consumed by operations such as task division will be lower. When the point size is small, the efficiency of one iteration in [65] will be higher than that in this paper. When the data point size is larger than a certain threshold, the efficiency of one iteration of this paper will be higher than that of the literature [65]. In the solution, as the data scale becomes larger and larger, the efficiency advantage of the scheme in this paper will become more and more obvious. The experimental results are a good demonstration of the point of view. At the same time, the experimental results show that the threshold of the data point size is about 5,000 data points. When the data size is larger than 7000, the paper has less time to consume in one iteration. When the data size is less than 5,000, In the literature [65], the scheme consumes less time in one iteration. The time-consuming pairs of the two schemes are shown in Table 4.6.

In one iteration, we are not only concerned with the time consumption in this iteration but also hope that in each iteration, server C can take on more tasks and have a higher consumption time ratio. As the size of the data increases, such programs will become more efficient. For the client, the main thing to do is the encryption and decryption operations. In both operations, the number of encryption and decryption of the client is basically the same. However, in the [65] scheme, the ciphertext distance calculation and the ciphertext distance comparison size are

Table 4.7: Time of each participant in one iteration of the proposed paper

data size	C(ms)	P_1(ms)	P_2(ms)
500	20,923	385	354
1,000	23,296	747	691
2,000	24,381	1,501	1,328
5,000	24,639	3,564	3,276
10,000	31,618	6,301	6,247

Table 4.8: Time of each participant in one iteration of [65]

data size	C(ms)	P_1(ms)	P_2(ms)
500	12,503	294	304
1,000	19,370	327	348
2,000	25,357	412	426
5,000	31,076	621	607
10,000	49,814	652	658

improved Liu encryption, all operations of the encryption are linear operations, and the scheme in this paper adopts the Paillier encryption algorithm. The decryption of the algorithm requires exponential and modular operations on the group. For clients with less computing power, the improved Liu encryption algorithm should take less time than the Paillier encryption used in this paper. Therefore, theoretically, under the same-scale data set, the time spent by the client in the [65] will be lower than the time consumed by the client in the scheme of this paper. As the size of the data increases, the time spent in one iteration of this paper is relatively small, and the time consumed by the client is relatively large. Therefore, when the data size gets larger and larger, the client time consumption in this paper scheme is more and more large, and the occupation time of the server is relatively smaller. The time consumption of each participant in one iteration of the two schemes is shown in Table 4.7 and Table 4.8.

Finally, the time of k-means clustering algorithm with privacy protection and the classic k-means algorithm in one iteration is given. It can be seen that the time consumption caused by encryption is relatively large. However, as the size of the data increases, the ratio of the time consumption of an iteration to the classic k-means time consumption gets smaller and smaller. The time spent on this paper and the classic k-means algorithm in one iteration is shown in Table 4.9. The decryption time consumption of this paper is shown in Table 4.10.

Table 4.9: Time comparison in one iteration

data size	encryption(ms)	no encryption(ms)
500	23,872	7
1,000	25,095	7
2,000	25,572	9
5,000	32,640	24
10,000	42,746	50

Table 4.10: Time of decryption of the proposed protocol

data size	3-dimension(ms)	7-dimension(ms)
500	93	111
1,000	149	278
2,000	169	294
5,000	352	760
10,000	629	1,443

4.4 Potential Applications

The proposed privacy-preserving k-means clustering protocol has significant potential for applications in various scenarios where sensitive data needs to be analyzed collaboratively without compromising privacy. Some representative use cases include:

- **Large-Scale Voting Statistics:** In large-scale elections or surveys, analyzing voting patterns or producing statistical results often requires collaboration among multiple regions or organizations. Privacy-preserving clustering ensures that individual votes and sensitive data remain confidential while enabling accurate statistical analysis.

- **Healthcare and Medical Research:** Hospitals and research institutions can collaborate on analyzing patient records to identify disease patterns or classify patients into risk groups without compromising the confidentiality of individual records.

- **Financial Services:** Banks and financial institutions can leverage the protocol to perform collaborative analyses such as fraud detection, customer segmentation, and credit risk assessment while maintaining data privacy.

- **Cross-Industry Collaboration:** Organizations from different sectors, such as supply chain management or smart city planning, can jointly analyze data without exposing proprietary or personal information.

These examples demonstrate the versatility of the protocol in addressing privacy concerns while enabling secure and meaningful collaboration. Further research can focus on adapting the protocol to specific domain challenges and optimizing its efficiency in large-scale applications.

Chapter 5

Fluid MPC

In this chapter, we propose a fluid MPC protocol that supports dynamic participation. In this work, we study MPC with dynamically evolving parties in the dishonest majority setting. This gives much stronger security guarantees since we only require that in any given round of the computation, there is at least one honest party taking part. However, it is also more challenging than the honest majority. We now elaborate on our contributions.

- In this paper, we propose a multi-party computation called dynamic SPDZ, which supports a dishonest majority secure model. In addition, the set of parties involved during the execution could be changed.
- We first propose a 1-to-n oblivious linear evaluation protocol in the all but one dishonest majority secure model based on the 1-to-1 oblivious linear evaluation protocol structured from lattice learning with error.
- Compared to other fluid MPC protocols, the computation and communication costs are extremely low.

5.1 Protocol Overview

The whole protocol serves as a client-server model. We regard the parties who hold private input as clients and the parties who carry out computation tasks as servers.

The protocol is divided into four parts: preprocessing, input, execution, and output. The execution stage is divided into epochs. Each epoch includes two phases: the computation phase and the hand-off phase.

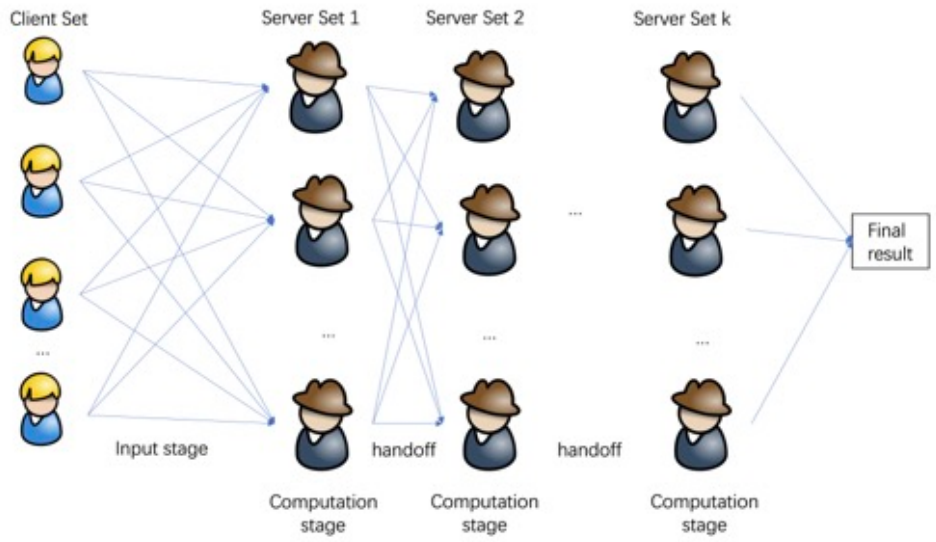

Figure 5.1: Framework of fluid MPC protocol

5.1.1 Secure Model

To effectively model fluid Multi-party Computation (MPC), we employ the arithmetic black box model (ABB), represented as an ideal functionality F_{ABB} within the universal composability framework. This functionality enables a set of parties, $P_1, \ldots, P_n$, to input values. The set of parties P_{set_j} execute computations and retrieve outputs. Crucially, F_{ABB} is parameterized by a finite field $\mathbb{F}_p$, and inherently supports the native operations of addition and multiplication within this field.

The F_{DABB} functionality thus provides a robust framework for multiparty computations,

ensuring that all operations are conducted securely, accurately, and efficiently in a distributed environment. This protocol involves a set of parties, denoted as $P_{\text{init}} = \{P_1, \ldots, P_n\}$, who engage in a series of computational steps, each governed by well-defined parameters and procedures.

Central to this functionality is the process of initialization, where the initial set of parties, P_{init}, is established as the client set for the commencement of computations. As the protocol progresses, it adeptly handles inputs from various parties, ensuring the accurate and secure incorporation of each participant's data. A dynamic transition mechanism updates the current set of active parties, P_{curr}, facilitating flexible and adaptive participation throughout the computation process.

The core of F_{DABB} lies in its ability to perform fundamental arithmetic operations, such as addition and multiplication, on the inputs provided. The results of these operations are meticulously stored, laying the groundwork for subsequent steps or outputs. In the final stage, the functionality is equipped to output the results of the computation, involving a meticulous process of retrieving and delivering these results to either the participating parties or an adversary, based on specific conditions, thereby maintaining the integrity and confidentiality of the entire process.

Overall, the F_{DABB} functionality offers a robust and versatile framework for multiparty computations, ensuring that all activities, from data input to result output, are conducted with utmost security, accuracy, and efficiency in a distributed computing environment.

Functionality F_{DABB}

Parameters: Finite field $\mathbb{F}_p$, a set of parties $P_{init} = \{P_1, ..., P_n\}$ which hold private input. Sever sets of server parties denote $P_{set_1}, P_{set_1}, ..., P_{set_k}$ carry out computation task. All parties have agreed upon identifiers id_x for each variable x used in the computation.

Initialise: On input $(Init, P_{init})$ from every $P_i \in P_{init}$ and set P_{init} as client set, $P_{curr} := P_{init}$.

Input: On input $(Input, id_x, x)$ from $P_i \in P_{init}$, and $(Input, id_x)$ from all other parties in P_{init} , store the pair (id_x, x).

Trans: On input $(Trans, P_{curr}, P_{set_k})$ from all $P_i \in P_{curr} \cup P_{set_k}$, update $P_{curr} := P_{set_k}$.

Add: On input (Add, id_z, id_x, id_y) from P_i , for every $P_i \in P_{curr}$, compute $z = x + y$ and store (id_z, z).

Multiplication: On input $(Mult, id_z, id_x, id_y)$ from every $P_i \in P_{curr}$, compute $z = x \cdot y$ and store (id_z, z).

Output: On input $(Output, id_z)$ from every $P_i \in P_{curr}$, where id_z has been stored previously, retrieve (id_z, z) and send it to the adversary. Wait for input from the adversary, if it is Deliver, send the output to every $P_i \in P_{curr}$. Otherwise, abort.

5.2 Preprocessing Phase for Dynamic Committees

5.2.1 Preprocessing Functionality

Firstly, we give the definition of the Functionality F_{prep} protocol, a sophisticated cryptographic framework designed for secure distributed computing within a finite field $\mathbb{F}_p$. The protocol involves key participants, namely a set of initiating parties, $P_{\text{init}} = \{P_1, P_2, \ldots, P_n\}$, holding private inputs, and multiple server party sets, P_{set_k}, responsible for processing and securely

exchanging these inputs.

The protocol's primary functions include:

- **Initialization**: Generating and distributing a unique MAC key for each participant to authenticate communications and computations.
- **Input Random Process**: Securely sampling and distributing random values among parties, maintaining data integrity.
- **Inner Random Process**: Generating a collective random value from individual random inputs of the parties, crucial for randomness in distributed computations.
- **Inner Triple Process**: Generating authenticated random triples, where each party holds a part of the triple and the sum forms the actual triple values.
- **Transfer Random Process**: Facilitating the secure transfer of random values between different server party sets, preserving data security and integrity.

Overall, the Functionality F_{prep} protocol is a cornerstone for secure, authenticated, and distributed computations, enabling collaborative computing over shared data without compromising security and privacy.

Functionality F_{prep}

Parameters: Finite field $\mathbb{F}_p$. Parties $P_{init} = \{P_1, P_2, ..., P_n\}$ who hold the private input. $P_{set_k}, (k \in [1, m])$, are m sets of server parties.

Functionality: Generate authenticated random triples and authenticated random values used for different stages.

Init: When receiving **Init** from all P_i where $P_i \in P_{init} \cup P_{set_k}$, generate a MAC key $\alpha^i \leftarrow \mathbb{F}_p$ for P_i and sends it to P_i.

Input Random: On input $(InPut, P_i, P_{set_1})$ from P_i, and $(InPut, P_i, P_{set_1})$ every $P_j \in P_{set_1}$,

1. Sample $r_i \leftarrow \mathbb{F}_p$ for P_i.

2. For each $P_j \in P_{set_1}$, sample $K_j^i \leftarrow \mathbb{F}_p$, and compute $M_i^j = r_i \cdot \alpha_j + K_j^i$, where α_j is the MAC key belongs to P_j.

3. Return r_i, M_i^j to P_i, and K_j^i to P_j.

Inner Random : On input $(InRand, P_{set_k})$ from all $P_i \in P_{set_k}$,

1. Sample $r_i \leftarrow \mathbb{F}_p$ for $P_i \in P_{set_k}$.

2. Compute $r = \sum r_i$, $R = r \cdot \sum \alpha_i$, where α_i is the MAC key belongs to P_i.

3. Sample $R_i \leftarrow \mathbb{F}_p$ for $P_i \in P_{set_k}$ such that $R = \sum R_i$.

4. Return $[\![r]\!] = \{r_i, R_i\}$ to P_i.

Inner Triple: On input $(InTriple, P_{set_k})$ from all $P_i \in P_{set_k}$,

1. Sample $a_i, b_i \leftarrow \mathbb{F}_p$ for $P_i \in P_{set_k}$.

2. Compute $a = \sum a_i, b = \sum b_i, c = a \cdot b$,

3. Compute $A = a \cdot \sum \alpha_i$, $B = b \cdot \sum \alpha_i$, $C = c \cdot \sum \alpha_i$ where α_i is the MAC key belongs to P_i.

3. Sample $c_i, A_i, B_i, C_i \leftarrow \mathbb{F}_p$ for $P_i \in P_{set_k}$, where $c = \sum c_i$, $A = \sum A_i$, $B = \sum B_i$, $C = \sum C_i$.

4. Return $[\![a]\!] = \{a_i, A_i\}, [\![b]\!] = \{b_i, B_i\}, [\![c]\!] = \{c_i, C_i\}$ to P_i.

Transfer Random : On input $(TrRand, P_{set_k}, P_{set_{k+1}})$ from all $P_i \in P_{set_k}$, and $P_j \in P_{set_{k+1}}$:

1. Sample $r_i \leftarrow \mathbb{F}_p$ for $P_i \in P_{set_k}$.
2. Compute $r = \sum r_i$, $R = r \cdot \sum \alpha_i$, where α_i is the MAC key belongs to P_i.
3. Sample $R_i \leftarrow \mathbb{F}_p$ for $P_i \in P_{set_k}$, where $R = \sum R_i$.
4. For each $P_j \in P_{set_k}$, sample $r'_j, R'_j \leftarrow F_p$, such that $\sum r'_j = r$, $\sum R'_j = r \cdot \sum \alpha_j$.
5. Sample $M_i, K_j \leftarrow \mathbb{F}_p$ for $P_i \in P_{set_k}$ and $P_j \in P_{set_{k+1}}$, such that, $Z = \sum M_i + \sum K_j$, where α_j is the MAC key belongs to P_j.
6. Return $[\![r]\!]^{P_{set_k}} = \{r_i, R_i\}$ to P_i, and $[\![r]\!]^{P_{set_{k+1}}} = \{r'_i, R'_i\}$ to P_j.

5.2.2 Preprocessing Protocol

In order to realize the F_{prep}, we introduce two building blocks: a $1-n$ oblivious linear evaluation function (F_{1-nOLE}) and a $n-n$ oblivious linear evaluation (F_{nOLE}). we elaborate on these below and show how they can be realized.

Firstly, we give the definition of F_{1-nOLE}. The F_{1-nOLE} functionality is a critical component in cryptographic computations, particularly designed for secure operations in a distributed environment. Operating within a finite field $\mathbb{F}_p$, this functionality involves a specific party P_i and a set of parties P_{set_1}, each holding a unique MAC key for authentication purposes.

The core operation, termed as *Extend*, is initiated by P_i and entails a series of computations and exchanges with the parties in P_{set_1}. This process includes sampling of random values and generating authenticated messages, ensuring both the secrecy and integrity of the exchanged data. The F_{1-nOLE} functionality is thus instrumental in extending the capability of the system to handle secure, authenticated, and distributed computations effectively.

Functionality F_{1-nOLE}

Parameters: Finite field $\mathbb{F}_p$. A party P_i and a set of parties P_{set_1}. Each party $P_j \in P_{set_1}$ holds a MAC key α_j.

Extend: On receiving $(Extend, P_i, P_{set_1})$ from P_i, and $(Extend, P_i, P_{set_1}, \alpha_j)$ from every $P_j \in P_{set_1}$, execute the following construct:

1. Sample $r_i \leftarrow \mathbb{F}_p$ for P_i.

2. For each $P_j \in P_{set_1}$, sample $K_j^i \leftarrow \mathbb{F}_p$, and compute $M_i^j = r_i \cdot \alpha_j + K_j^i$, where α_j is the MAC key belongs to P_j.

3. Return r_i, M_i^j to P_i, and K_j^i to P_j.

Secondly, the protocol of the preprocessing phase is also built based on the following function, called F_{VOLE}. The *Functionality* F_{nOLE} involves a set of operations within a finite field $\mathbb{F}_p$. It includes two distinct sets of parties: P_{set_k} and $P_{set_{k+1}}$. Each party in the second set, denoted as P_j where $P_j \in P_{set_{k+1}}$, is assigned a unique MAC (Message Authentication Code) key α_j.

The core operation of this functionality is defined as the 'Extend' process. This process is triggered when specific inputs are received from the parties in both P_{set_k} and $P_{set_{k+1}}$. Specifically, the 'Extend' operation commences upon receiving inputs in the form of $(Extend, P_{set_k}, P_{set_{k+1}}, x_i)$ from all parties $P_i \in P_{set_k}$, and $(Extend, P_{set_k}, P_{set_{k+1}}, \alpha_j)$ from all parties $P_j \in P_{set_{k+1}}$.

Functionality F_{nOLE}

Parameters: Finite field $\mathbb{F}_p$. Two sets of parties $P_{set_k}, P_{set_{k+1}}$. Each party $P_j \in P_{set_{k+1}}$ holds a MAC key α_j.

Extend: On receiving $(Extend, P_{set_k}, P_{set_{k+1}}, x_i)$ from all $P_i \in P_{set_k}$, and $(Extend, P_{set_k}, P_{set_{k+1}}, \alpha_j)$ from all $P_j \in P_{set_{k+1}}$, execute the following construct:

1. For each $P_i \in P_{set_k}$, $P_j \in P_{set_{k+1}}$, sample $K_j^i \leftarrow F_p$ and compute $M_i^j = x_i \cdot \alpha_j$.
2. Return M_i^j to P_i, and K_j^i to P_j.

Finally, we will show our protocol π_{Prep}, which UC-secure utilizes F_{prep}.

The Π_{Prep} protocol is a sophisticated framework for secure multi-party computations in $\mathbb{F}_p$. It involves initial parties $P_{\text{init}} = \{P_1, P_2, \ldots, P_n\}$ holding private inputs, divided into n sets P_{set_i}, $i \in [1, n]$. The protocol progresses through several stages:

- **Initialization:** Sets up the protocol with the assumption of sufficient random numbers and triples for the computation.

- **Inner Random Values Setup:** Generates shared random values $[\![r]\!]$ within each P_{set_k}, involving cryptographic interactions for secure computation.

- **Inner Triples Setup:** Produces triples $[\![a]\!], [\![b]\!], [\![c]\!]$ where $c = a \cdot b$, based on previously generated values.

- **Transfer Random Value Setup:** Creates random values for data transfer between P_{set_k} and $P_{\text{set}_{k+1}}$.

- **Input Random Value Setup:** Generates random values for sharing private inputs from P_i to P_{set_1}.

The protocol includes functions for retrieving inner random values, inner triples, transfer

random values, and input random values, which are crucial for the secure and efficient execution of MPC.

Protocol Π_{Prep}

Parameters: Finite field $\mathbb{F}_p$. Parties $P_{init} = \{P_1, P_2, ..., P_n\}$ who hold the private input. $P_{set_i}, (i \in [1, n])$ are n sets of server parties.

Init: run the following step among all parties, suppose m random numbers and triples are enough to support the whole computation.

Inner Random values setup: To generate $[\![r]\!]$ among P_{set_k},

1. For all $P_i \in P_{set_k}$ generate a random value $r_i \leftarrow F_p$.

2. P_i call F_{nOLE} with input $(Extend, P_{set_k}, P_{set_k}, x_i)$ and $(Extend, P_{set_k}, P_{set_k}, \alpha_i)$.

3. On receiving K_i^j, M_i^j, P_i computes $R_i = \sum(M_i^j + K_i^j)$, where $R = \sum R_i = \sum r_i \cdot \sum \alpha_i = r \cdot \alpha_{set_k}$. This format $[\![r]\!] = [r], [R]$

Inner Triples setup: To generate $[\![a]\!], [\![b]\!], [\![c]\!]$ among P_{set_k}, where $c = a \cdot b$,

1. Suppose P_{set_k} already hold $[\![a]\!], [\![b]\!]$ from *Inner Random values setup* stage.

2. $P_i \in P_{set_k}$ call F_{nOLE} with input $(Extend, P_{set_k}, P_{set_{k+1}}, a_i)$ and $P_j \in P_{set_{k+1}}$ call F_{nOLE} with input $(Extend, P_{set_k}, P_{set_{k+1}}, b_j)$.

3. On receiving K_i^j, M_i^j, P_i computes $c_i = \sum(M_i^j + K_i^j)$, where $c = \sum c_i = \sum a_i \cdot \sum b_i = a \cdot b$. This format $[c]$.

4. For each $P_i \in P_{set_k}$ call F_{nOLE} with $(Extend, P_{set_k}, P_{set_k}, c_i)$, and $(Extend, P_{set_k}, P_{set_k}, \alpha_j)$.

5. On receiving K_i^j, M_i^j, P_i computes $C_i = \sum(M_i^j + K_i^j)$, where $C = \sum C_i = \sum c_i \cdot \sum \alpha_i = c \cdot \alpha_{set_k}$. This format $[c \cdot \alpha_{set_k}]$.

Transfer Random Value setup: To generate random values used for transferring data between P_{set_k} and $P_{set_{k+1}}$,

1. Suppose P_{set_k} already hold $[\![r]\!]^{set_k}$ from *Inner Random values setup* stage. $P_i \in P_{set_k}$ call F_{nOLE} with input $(Extend, P_{set_k}, P_{set_{k+1}}, r_i)$ and $P_j \in P_{set_{k+1}}$ call F_{nOLE} with input $(Extend, P_{set_k}, P_{set_{k+1}}, \alpha_j)$.

2. On receiving M_i^j, P_i computes $M_i = \sum M_i^j$. On receiving K_j^i, P_j computes $K_j = \sum K_j^i$.

Input Random Value setup: To generate random value used for sharing private input of P_i to P_{set_1},

1.P_i generate a random $r_i \leftarrow F_p$, P_i call F_{1-nOLE} with input $(Extend, P_i, P_{set_1}, r_i)$ and all $P_j \in P_{set_1}$ with input $(Extend, P_i, P_{set_1}, \alpha_j)$. And then, P_i receives M_i^j and P_j receives K_j^i, where $M_i^j = r_i \cdot \alpha_j + K_j^i$.

Inner Random: On receiving $(InRandom, P_{set_k})$ from all $P_i \in P_{set_k}$,

1. Let $[\![r]\!]$ be the secret sharing generated in *Inner Random values setup* stage, and has not been used before.

2. Return r_i, R_i to P_i.

Inner Triple: On receiving $(InTriple, P_{set_k})$ from all $P_i \in P_{set_k}$,

1. Let $[\![a]\!], [\![b]\!], [\![c]\!]$ be the triple generated in *Inner Triples setup* stage, and has not been used before.

2. Return $a_i, b_i, c_i, A_i, B_i, C_i$ to P_i.

Transfer Random: On receiving $(TrRandom, P_{set_k}, P_{set_{k+1}})$ from all $P_i \in P_{set_k}$, and $(TrRandom, P_{set_k}, P_{set_{k+1}})$ from all $P_j \in P_{set_{k+1}}$,

1. Return $[\![r]\!]$, M_i to P_i, and K_j to P_j.

Input Random: On receiving $(InRandom, P_i, P_{set_1})$ from P_i and all $P_j \in P_{set_1}$,

1. Return r_i, M_i^j to P_i, and K_j^i to P_j.

5.2.3 Instantiating Multi-Party OLE

In this section, we will show how to realize two different kinds of oblivious linear evaluation called Π_{1-nOLE} and Π_{nOLE}.

The Π_{1-nOLE} protocol, as detailed in this document, is a comprehensive cryptographic scheme designed to facilitate secure and verified computations within a finite field $\mathbb{F}_p$. This protocol involves a primary party, P_i, and a set of parties, P_{set_1}, each possessing a private MAC key, α_j, crucial for ensuring the integrity and confidentiality of the computations.

At the heart of the Π_{1-nOLE} protocol is the *Extend* function, which is triggered upon specific requests from the parties involved. The process involves the generation of random numbers, secure interactions between pairs of parties, and a series of consistency checks to guarantee the integrity of inputs across the protocol.

The protocol is designed with a focus on ensuring that all parties involved can verify the consistency and authenticity of the computations, making it an essential tool in environments where secure multiparty computation is required. It carefully balances the need for security with the efficiency of cryptographic operations, ensuring that the protocol is both robust and practical for real-world applications.

Protocol Π_{1-nOLE}

Parameters: Finite field $\mathbb{F}_p$. A party P_i and a set of parties P_{set_1}, Each party $P_j \in P_{set_1}$ hold private mac key α_j.

Extend: On receiving $(Extend, P_i, P_{set_1})$ from P_i, and $(Extend, P_i, P_{set_1}, \alpha_j)$ from all $P_j \in P_{set_1}$.

1. P_i generate a random number $r_i \leftarrow F_p$.

2. Each pair of parties (P_i, P_j), where $P_j \in P_{set_1}$ call F_{OLE} with P_i input $(Extend, P_i, P_j, r_i)$ and P_j input $(Extend, P_i, P_j, \alpha_j)$. And P_i gets M_i^j and and P_j gets K_j^i.

3. Consistency check: All parties need to complete the consistency check to guarantee P_i input the same r_i:

(a) All parties belong to $P_i \cup P_{set_k}$ generate a sequence of random numbers $\lambda_1, ..., \lambda_m$, and for $P_j \in P_{set_1}$ compute $Z_j = \lambda_j \cdot K_j^i$. P_j rerandomizes K_j^i locally by sending a zero share to the other parties, and P_j gets K_j. P_j broadcast K_j and compute $K = \sum K_j$.

(b) P_i compute $M = \sum \lambda_j \cdot M_i^j$, $Z = (M - K)^{-1}$. P_i generate $Z = \sum Z_j$ and sends Z_j to P_j secretly.

(c) $P_j \in P_{set_k}$ computes $Y_j = Z_j - \lambda_j \cdot \alpha_j$. P_j rerandomizes Y_j locally by sending a zero share to the other parties, and P_j gets Y_j'. P_j broadcast Y_j', compute $Y = \sum Y_j'$, and check $Y = 0$. If the check fails, abort.

The Π_{nOLE} protocol, as described in this document, is a robust cryptographic mechanism designed to facilitate secure and efficient computations within a finite field $\mathbb{F}_p$. The protocol involves two sets of parties, P_{set_k} and $P_{\text{set}_{k+1}}$, with each party in $P_{\text{set}_{k+1}}$ possessing a private MAC key, α_j, essential for the authentication and integrity of the computations.

Central to the Π_{nOLE} protocol is the *Extend* function, which is activated upon receiving spe-

cific inputs from all parties in both sets. This function includes a series of pairwise operations between the parties, leading to the generation of key values and a comprehensive consistency check. The protocol ensures that each party inputs consistent data through a sequence of calculated broadcasts and local computations.

The Π_{nOLE} protocol is tailored for environments where secure multiparty computation is critical. Its design emphasizes the verification of the authenticity and consistency of the computational inputs and outputs, thereby serving as a key tool in distributed computing scenarios where data integrity and security are paramount.

Protocol Π_{nOLE}

Parameters: Finite field $\mathbb{F}_p$. Two sets of parties $P_{set_k}, P_{set_{k+1}}$. Each party $P_j \in P_{set_{k+1}}$ holds a MAC key α_j.

Extend: On receiving $(Extend, P_{set_k}, P_{set_{k+1}}, x_i)$ from all $P_i \in P_{set_k}$, and $(Extend, P_{set_k}, P_{set_{k+1}}, \alpha_j)$ from all $P_j \in P_{set_{k+1}}$,

1. For each pair (P_i, P_j) where $P_j \in P_{set_{k+1}}$ call F_{OLE} with P_i input $(Extend, P_i, P_j, x_i)$ and P_j input $(Extend, P_i, P_j, \alpha_j)$. And P_i gets M_i^j and and P_j gets K_j^i.

2. Consistency check: All parties need to complete the consistency check to guarantee P_i input the same r_i and P_j input the same α_j:

(a) All parties belong to $P_{set_k} \cup P_{set_{k+1}}$ generate a sequence of random numbers $\lambda_1, ..., \lambda_m$, and for $P_i \in P_{set_k}$ compute $M_i = \sum \lambda_j \cdot M_i^j$. P_i broadcasts M_i and compute $M = \sum M_i$.

(b) $P_j \in P_{set_{k+1}}$ compute $K_j = \sum \lambda_j \cdot K_i^j$ locally. P_j broadcasts K_j and computes $k = \sum K_j$.

(c) P_i rerandomizes r_i locally by sending a zero share to the other parties, and P_i gets r_i'. P_i broadcasts r_i' and computes $r = \sum r_i'$.

(d) P_j computes $Z_j = \lambda_j \cdot \alpha_j$. P_j rerandomizes Z_j locally by sending a zero share to the other parties, and P_j gets Z_j'. P_j broadcasts Z_j' and computes $Z = \sum Z_j'$.

(e) All party check $Z \cdot r + M = K$. If the check fails, abort.

5.3 Online Stage

5.3.1 Building Blocks for Online Stage

In this section, we describe the online stage of the dynamic SPDZ protocol. Before introducing the online stage, we first introduce two protocols called $\Pi_{key-switch}$ and $\Pi_{Mac-check}$.

The protocol section described involves a switch mechanism for transferring a shared value $[\![x]\!]$ between two sets of parties, P_{set_k} and $P_{set_{k+1}}$, in a finite field F_p. The shared value $[\![x]\!]^{Pset_k}$ includes a value $[x]$ and a multiplication authentication code (MAC) $[\alpha_{set_k} \cdot x]$, with α_{set_k} being the sum of individual α_i values for each party P_i in $Pset_k$.

The process to switch $[\![x]\!]$ to $P_{set_{k+1}}$ is as follows:

Preparation Phase: Each party P_i in P_{set_k} initiates a call to a function F_{prep} with parameters $(TrRand, P_{set_k}, P_{set_{k+1}})$. As a result, P_i receives a part of the shared random value $[\![r]\!]^{P_{set_k}}$ specific to P_{set_k}, and similarly, each P_j in $P_{set_{k+1}}$ receives their part of $[\![r]\!]^{P_{set_{k+1}}}$.

Computation and Opening Phase: Parties in P_{set_k} collaboratively compute and then reveal the value of $[\![x + r]\!]^{P_{set_k}}$. Subsequently, both sets of parties, P_{set_k} and $P_{set_{k+1}}$, execute the $\Pi_{reshare}$ protocol to securely reshare $[x]$ from P_{set_k} to $P_{set_{k+1}}$.

Final Computation Phase: Each party P_j in $P_{set_{k+1}}$ computes their share of the MAC $[\alpha^{P_{set_{k+1}}} \cdot x]$ using the formula $[\alpha^{P_{set_{k+1}}}] \cdot (x + r) - [\alpha^{P_{set_{k+1}}} \cdot r]$. At the end of this process, $P_{set_{k+1}}$ collectively holds both the value $[x]$ and its associated MAC $[\alpha^{P_{set_{k+1}}} \cdot x]$.

This protocol section ensures the secure and verifiable transfer of a shared value and its MAC between two different sets of parties within a secure multi-party computation framework.

Protocol $\Pi_{key-switch}$

Parameters: Finite field $\mathbb{F}_p$. Two sets of parties $P_{set_k}, P_{set_{k+1}}$. $[\![x]\!]^{P_{set_k}} = ([x], [\alpha_{set_k} \cdot x])$. $\alpha_{set_k} = \sum \alpha_i$, where $P_i \in P_{set_k}$.

Switch: To get $[\![x]\!]^{P_{set_{k+1}}} = ([x], [\alpha_{set_{k+1}} \cdot x])$. $\alpha_{set_{k+1}} = \sum \alpha_j$, where $P_j \in P_{set_{k+1}}$.

1. Each $P_i \in P_{set_k}$ call F_{prep} with $(TrRand, P_{set_k}, P_{set_{k+1}})$, $P_j \in P_{set_k}$ call F_{prep} with $(TrRand, P_{set_k}, P_{set_{k+1}})$. P_i receives $[\![r]\!]^{P_{set_k}} = \{r_i, R_i\}$, and P_j receives $[\![r]\!]^{P_{set_{k+1}}} = \{r'_j, R'_j\}$.

2. Parties in P_{set_k} compute and open $[\![x + r]\!]^{P_{set_k}}$. Parties in $P_{set_k} \cup P_{set_{k+1}}$ run $\Pi_{reshare}([x]^{P_{set_k}}, P_{set_k}, P_{set_{k+1}})$ to get $[x]^{P_{set_{k+1}}}$.

3. Finally, P_j can compute its share of the MAC $[\alpha^{P_{set_{k+1}}} \cdot x]$ as $[\alpha^{P_{set_{k+1}}}] \cdot (x + r) - [\alpha^{P_{set_{k+1}}} \cdot r]$. $P_{set_{k+1}}$ holds $[x], [\alpha^{P_{set_{k+1}}} \cdot x]$.

The protocol section titled $\Pi_{Mac-check}$ outlines a procedure used by a set of parties, denoted as P_{set_k}, to verify the integrity of Multiplication Authentication Codes (MACs) on a series of values $(a_1, a_2, ..., a_m)$. Each party in P_{set_k} possesses a share of the product of each value a_j and a collective key α_{set_k}, denoted as $A_{j,i}$. Here's an explanation of how the MAC check is performed:

Random Number Generation: All parties first generate a sequence of random numbers $(r_1, r_2, ..., r_m)$. These random numbers are crucial for ensuring the randomness and unpredictability of the verification process.

Public Value Computation: Each party computes a public value a, which is the sum of the products of each random number r_j with the corresponding value a_j (i.e., $a = \sum r_j \cdot a_j$).

Individual Computation and Broadcasting: For each party P_i in P_{set_k}, they compute a value K_i which is the sum of the products of each random number r_j with their share of the MAC for a_j (i.e., $K_i = \sum r_j \cdot A_{j,i}$). Subsequently, they compute $M_i = K_i - \alpha_i \cdot a$ and broadcast M_i to

all other parties.

Verification and Abort Condition: After receiving the broadcasted values from each party, the parties sum up these values (i.e., $M_1 + M_2 + ... + M_n$). If the sum is not equal to zero, it indicates a discrepancy in the MACs, and the parties abort the protocol.

This MAC check protocol is a crucial aspect of secure multi-party computation, as it ensures the integrity and authenticity of shared values among parties, preventing malicious activities or errors in the computation process.

Protocol $\Pi_{Mac-check}$

Usage: Parties in P_{set_k} want to check the MACs on values $(a_1, a_2, ..., a_m)$ opened to them. Each $P_i \in P_{set_k}$ holds share of $a_j \cdot \alpha_{set_k}$ denotes $A_{j,i}$.

$MACCheck(a_1, ..., a_t)$:

1. All parties get a sequence of random numbers $r_1, ..., r_m$.
2. Each party computes the public value $a = \sum r_j \cdot a_j$
3. For each party P_i, P_i computes $K_i = \sum r_j \cdot A_{j,i}$, $M_i = K_i - \alpha_i \cdot a$. P_i broadcast M_i
4. If $M_1 + ... + M_n \neq 0$, the parties abort.

The protocol Π_{share} is a sub-protocol of $\Pi_{key-swith}$, which is used to share a value x_i to a set of parties in additive sharing format.

Protocol $_{share}$

Parameters: Finite fields $\mathbb{F}_q$. Party P_i with private input x_i and a set of parties P_{set_1}. Each pair of parties (P_i, P_j) , where $P_j \in P_{set_1}$, has a common PRG seed $S^{i,j}$. Suppose $P_{set_1} = \{P_1, P_2, ..., P_m\}$.

Functionality: To get $[x_i]^{P_{set_1}} = \{x_{i,j}\}_{P_j \in P_{set_1}}$

Share:

1. P_i computes $x_{i,j} \leftarrow PRG(S^{i,j})$, for $j = 2, 3, ..., m$. P_i defines $x_{i,1} = x_i - \sum_{j=2}^{m} x_{i,j}$.

2. P_i sends $x_{i,1}$ to $P_1 \in P_{set_1}$. Each $P_j \in P_{set_1}$ defines its shares as $[x_i] = x_{i,j}$ which forms $[x_i]^{P_{set_1}}$.

5.3.2 Protocol of Online Stage

The online stage includes Input, Computation, Hand-off and output phases. The Π_{online} protocol is a comprehensive framework designed for secure multi-party computation (MPC) in a finite field $\mathbb{F}q$. It involves a set of clients, $Pinit$, each with a private input, and several sets of server parties, P_{set_k}, who execute the computation tasks in different stages. Each server party also possesses a private MAC key, α_j. Here's an overview of the protocol stages:

Input Sharing:

Each client P_i and server party P_j in P_{set_1} call a preparation function F_{prep}. Clients compute and share their inputs with the server parties. The shared inputs are in the format $[\![\cdot]\!]$, which includes both the input value and its MAC. Server parties perform calculations to obtain the MAC of the random value associated with each input. Computation Phase:

The protocol supports four basic operations: addition, addition by constant, multiplication by constant, and multiplication. For addition, parties locally add their shares of the input values. In addition by constant, a designated party modifies the shared value by adding the constant, and

all parties adjust their MAC shares accordingly. For multiplication by constant, each party scales their input share and corresponding MAC share by the constant. Multiplication of two values is handled by a dynamic multiplication protocol, $\Pi_{dynamic-Mult}$. Hand-off Between Server Sets:

This process involves transferring the computation from one set of server parties to another, P_{set_k} to $P_{set_{k+1}}$. They utilize a key-switching protocol, $\Pi_{key-switch}$, to ensure that the new set of parties correctly receives the shared values along with their updated MACs. Output Generation:

The final stage involves generating the output of the computation, which is not elaborated upon in the provided description. This protocol is designed to ensure privacy and integrity of computations in a multi-party setting, leveraging MACs for authenticity and supporting a variety of operations fundamental to MPC.

Protocol Π_{online}

Parameter: Finite field $\mathbb{F}_q$. Initially, each data owner called client $P_i \in P_{init}$ has a private input x_i. Several sets of server parties denote P_{set_k}, which carry out the computation task in different stages. In addition, each server party P_j owns a private mac key α_j

Input: To share input x_i belongs to user $P_i \in P_{init}$ to a set of parties P_{set_1} in $[\![\cdot]\!]$ format :

1. For each P_i call F_{prep} with input $(InPut, P_i, P_{set_1})$ and $P_j \in P_{set_1}$ with input $(InPut, P_i, P_{set_1})$. And then, P_i receives r_i, M_i^j and P_j receives K_j^i, where $M_i^j = r_i \cdot \alpha_j + K_j^i$.

2. P_i computes $M_i = \sum M_i^j$, where $P_j \in P_{set1}$. And then P_i execute $\Pi_{share}(M_i)$ and $\Pi_{share}(x_i)$ with P_{set_1}. $P_j \in P_{set_1}$ receives $[M] = M_j^i$ and $[x_i] = x_j^i$. In addition, P_i public $Y = x_i + r_i$ to P_{set1}.

3. For each party $P_j \in P_{set1}$, received M_j^i from party $P_i \in P_{init}$. P_j computes $[\alpha \cdot r] = [M] - [K] = M_j^i - K_j^i$.

4. For each $P_j \in P_{set1}$ computes $[x_i \cdot \alpha] = \alpha_j \cdot Y - [\alpha \cdot r_i]$ which format $[\![x_i]\!]^{P_{init}}$

Computation: In the computation phase, the protocol supports 4 kinds of operations among a set of parties denoting P_{set_k}, including addition, addition by constant, multiplication by constant, and multiplication.

Addition: To execute the addition operation in the circuit, $z = x + y$, each $P_i \in P_{set_k}$ locally adds their share of x and y to get share of z, $[\![z]\!] = [\![x]\!] + [\![y]\!]$.

Addition by Constant: To execute the addition by constant operation, $z = x + c$, a designed party $P_j \in P_{set_k}$ adds c to $[x]$ to get $[x + c]$ and for all party $P_i \in P_{set_k}$ add $\alpha_i \cdot c$ to $[x \cdot \alpha]$ to get $[(x + c) \cdot \alpha]$.

Multiplication by Constant: To compute, $z = x \cdot c$, each $P_i \in P_{Set_k}$ locally compute $[x \cdot c] = [x] \cdot c$, $[\alpha \cdot (x \cdot c)] = [\alpha \cdot x] + \alpha_i \cdot c$

Multiplication: To compute, $z = x \cdot y$, run $\Pi_{dynamic-Mult}$ among P_{set_k}

Hand-off: There are two sets of parties $P_{set_k}, P_{set_{k+1}}$. Every party $P_i \in P_{set_k}$ hold shares $[\![x_u]\!] = \{[x_u], [x_u \cdot \alpha_{set_k}]\}$. The two sets of parties run $\Pi_{key-switch}$ and parties $P_j \in P_{set_{k+1}}$ receives $[\![x_u]\!] = \{[x_u], [x_u \cdot \alpha_{set_{k+1}}]\}$. In addition, the set of parties P_{set_k} also run $\Pi_{Mac-check}$. If $\Pi_{Mac-check}$ fails, rejects.

Output: To output the final result, for each output wire z, they open $[\![z]\!]$ by broadcasting their shares to the other parties and running $\Pi_{Mac-check}$. If $\Pi_{Mac-check}$ fails, rejects.

Protocol $\Pi_{fluid-Mult}$

Usage: P_{set_k} wants to compute multiplications $z = x \cdot y$

1. For every parties $P_i \in P_{set_k}$ call F_{prep} with $(InTriple, P_{set_k})$ and receives $[\![a]\!], [\![b]\!], [\![c]\!]$. P_i computes $[\![\epsilon]\!] = [\![x]\!] - [\![a]\!]$, $[\![\rho]\!] = [\![y]\!] - [\![b]\!]$.

2. P_i open $[\![\epsilon]\!], [\![\rho]\!]$ and get ϵ, ρ.

3. P_i compute $[\![x \cdot y]\!] = [\![c]\!] + \epsilon \cdot [\![b]\!] + \rho \cdot [\![a]\!] + \epsilon \cdot \rho$

5.4 Cost Analysis

In this section, we analyze the efficiency of our proposed protocol compared to the Le Mans fluid MPC protocol. To facilitate this comparison, we define the following parameters:

- L: The number of layers in the computation circuit.
- P: The total number of multiplication gates in the circuit.
- n: The total number of parties participating in the computation stage.
- k: The number of sets into which the n parties are divided.
- q: The number of parties within each set.
- m: The number of private input holders.

5.4.1 Cost in Le Mans Fluid MPC Protocol

In the Le Mans fluid MPC protocol, the computational costs are structured as follows:

- **Preprocessing Stage**: To generate a shared random value $\langle r \rangle$ among all $m + n$ parties, $(m + n)^2$ oblivious linear evaluations (OLEs) are required. Additionally, generating a random multiplication triple $\langle a \rangle, \langle b \rangle, \langle c \rangle$ (where $c = a \cdot b$) requires $3(m + n)^2$ OLEs.
- **Input Stage**: Each of the m private input holders incurs a cost of m random number sharings.
- **Hand-off Stage**: If each set contains q parties, each hand-off operation incurs a cost of q random number sharings.
- **Multiplication Gates**: For each multiplication gate, a random triple sharing is required.

5.4.2 Cost in Our Proposed Fluid MPC Protocol

Our protocol optimizes the costs through efficient random value sharing mechanisms:

- **Preprocessing Stage**: We define four types of random value sharing:
  1. Input random value sharing: Costs q OLEs.
  2. Inner random value sharing: Costs q^2 OLEs.
  3. Inner random triple value sharing: Costs $3q^2$ OLEs.
  4. Transfer random value sharing: Costs q^2 OLEs.

  The total preprocessing cost is $mq + q^3 + pq^2$.
- **Input Stage**: For m private input holders, the total cost is m random number sharings.
- **Hand-off Stage**: Each operation incurs a cost of q transfer random number sharings.
- **Multiplication Gates**: Each multiplication gate requires a random inner triple sharing.

5.4.3 Comparison and Analysis

The preprocessing cost for the Le Mans fluid MPC protocol is:

$$\text{Le Mans Fluid MPC Preprocessing Cost} = m(m+n)^2 + q(m+n)^2 + 3p(m+n)^2.$$

In contrast, our proposed protocol reduces the preprocessing cost to:

$$\text{Our Protocol Preprocessing Cost} = mq + q^3 + pq^2.$$

Efficiency Comparison

Let us consider a practical example with the following parameters:

- $m = 10$: Number of private input holders.
- $n = 20$: Number of participating parties.
- $q = 5$: Number of parties in each set.
- $p = 50$: Number of multiplication gates.

The costs are computed as follows:

- **Le Mans Fluid MPC Preprocessing Cost**:

$$10(10+20)^2 + 5(10+20)^2 + 3(50)(10+20)^2 = 9000 + 4500 + 135000 = 148500.$$

- **Our Protocol Preprocessing Cost**:

$$(10)(5) + (5)^3 + (50)(5^2) = 50 + 125 + 1250 = 1425.$$

5.4.4 Conclusion

The results clearly demonstrate that our proposed protocol significantly reduces the preprocessing cost, lowering it from 148500 to 1425 under the given parameters. This reduction is achieved through the efficient use of random value sharing mechanisms, making our protocol particularly suitable for large-scale secure computations. At the same time, the input sharing, hand-off, and multiplication gate costs remain comparable, ensuring that the overall efficiency is not compromised.

Chapter 6

Conclusions and Suggestions for Future Research

In this thesis, we addressed key privacy challenges in secure multi-party computation (MPC) across various computational scenarios. Our proposed protocols enhance both the practicality and security of collaborative computing among untrusted entities, eliminating the need for a trusted third party.

First, we introduced a decentralized e-voting system that integrates blockchain technology, smart contracts, linkable ring signatures, and threshold encryption. This design safeguards voter privacy and ensures the integrity of election results. An Ethereum private network implementation demonstrates feasibility in terms of cost and time efficiency.

Second, we proposed a two-party k-means clustering scheme for privacy-preserving data mining. By optimizing data encryption and leveraging cloud-based execution, the protocol efficiently handles $\mathcal{O}(k(m+n))$ rounds of interaction and addresses high computational and communication overheads. The scheme is validated under both semi-honest and malicious security models, underscoring its robust privacy guarantees.

Finally, we developed a fluid MPC protocol, extending the SPDZ protocol to accommodate dynamic participant involvement in complex computations. This protocol's minimal preprocessing requirements and adaptability significantly lower barriers for large-scale, resource-intensive tasks. Its security in an all-but-one dishonest majority model broadens applicability in diverse computational settings.

Overall, we not only tackled existing MPC issues but also established a foundation for further research into more efficient, secure, and adaptable privacy-preserving protocols. The innovations we presented in e-voting, collaborative data mining, and fluid MPC underscore the potential for real-world deployment and ongoing academic exploration, ultimately aiming to safeguard privacy in the digital age.

References

[1] Martin Abadi, Andy Chu, Ian Goodfellow, H. Brendan McMahan, llya Mironov, Kunal Talwar, and Li Zhang. Deep learning with differential privacy. In *2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 308–318, 2016.

[2] Ben Adida. Helios: Web-based open-audit voting. In *Usenix Security Symposium*, pages 335–348, 2008.

[3] Rakesh Agrawal and Ramakrishnan Srikant. *Privacy-preserving data mining*. ACM, 2000.

[4] Abdelrahaman Aly, Karl Cong, Daniele Cozzo, Marcel Keller, Emmanuela Orsini, Dragos Rotaru, Oliver Scherer, Peter Scholl, Nigel P Smart, Titouan Tanguy, et al. Scale–mamba v1. 14: Documentation. *Documentation. pdf*, 2021.

[5] Gökhan Arslan, Mustafa Tuncan, M Talat Birgonul, and Irem Dikmen. E-bidding proposal preparation system for construction projects. *Building and Environment*, 41(10):1406–1413, 2006.

[6] Man Ho Au, Joseph K Liu, Tsz Hon Yuen, and Duncan S Wong. Id-based ring signature scheme secure in the standard model. In *International Workshop on Security*, pages 1–16, 2006.

[7] Marshall Ball, Tal Malkin, and Mike Rosulek. Garbling gadgets for boolean and arithmetic circuits. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 565–577, 2016.

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

[9] Josh Benaloh and Dwight Tuinstra. Receipt-free secret-ballot elections (extended abstract). In *Twenty-Sixth ACM Symposium on Theory of Computing*, pages 544–553, 1994.

[10] Rikke Bendlin, Ivan Damgård, Claudio Orlandi, and Sarah Zakarias. Semi-homomorphic encryption and multiparty computation. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 169–188. Springer, 2011.

[11] Fabrice Benhamouda, Shai Halevi, and Tzipora Halevi. Supporting private data on hyperledger fabric with secure multiparty computation. *IBM Journal of Research and Development*, 63(2/3):3–1, 2019.

[12] Dan Bogdanov, Sven Laur, and Jan Willemson. Sharemind: A framework for fast privacy-preserving computations. In *Computer Security-ESORICS 2008: 13th European Symposium on Research in Computer Security, Málaga, Spain, October 6-8, 2008. Proceedings 13*, pages 192–206. Springer, 2008.

[13] Beyza Bozdemir, Sébastien Canard, Orhan Ermis, Helen Möllering, Melek Önen, and Thomas Schneider. Privacy-preserving density-based clustering. In *Proceedings of the 2021 ACM Asia Conference on Computer and Communications Security*, pages 658–671, 2021.

[14] Paul Bunn and Rafail Ostrovsky. Secure two-party k-means clustering. In *the 14th ACM conference on Computer and communications security*, pages 486–497. ACM, 2007.

[15] David Chaum. Blind signatures for untraceable payments. In *Advances in cryptology*, pages 199–203, 1983.

[16] David Chaum and Eugène Van Heyst. Group signatures. In *Workshop on the Theory and Application of of Cryptographic Techniques*, pages 257–265, 1991.

[17] David L Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Communications of the ACM*, 24(2):84–90, 1981.

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

[19] Sherman SM Chow, Joseph K Liu, and Duncan S Wong. Robust receipt-free election system with ballot secrecy and verifiability. In *NDSS*, volume 8, pages 81–94, 2008.

[20] Jeremy Clark and Aleksander Essex. Commitcoin: Carbon dating commitments with bitcoin. In *International Conference on Financial Cryptography and Data Security*, pages 390–398, 2012.

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

[22] Ronald Cramer, Ivan Damgård, and Jesper B Nielsen. Multiparty computation from threshold homomorphic encryption. In *Advances in Cryptology—EUROCRYPT 2001: International Conference on the Theory and Application of Cryptographic Techniques Innsbruck, Austria, May 6–10, 2001 Proceedings 20*, pages 280–300. Springer, 2001.

[23] Ronald Cramer, Rosario Gennaro, and Berry Schoenmakers. A secure and optimally efficient multi-authority election scheme. In *International Conference on Theory and Application of Cryptographic Techniques*, pages 103–118, 1997.

[24] Ivan Damgård, Kasper Damgård, Kurt Nielsen, Peter Sebastian Nordholt, and Tomas Toft. Confidential benchmarking based on multiparty computation. In *International Conference on Financial Cryptography and Data Security*, pages 169–187. Springer, 2016.

[25] Ivan Damgård, Martin Geisler, Mikkel Krøigaard, and Jesper Buus Nielsen. Asynchronous multiparty computation: Theory and implementation. In *International workshop on public key cryptography*, pages 160–179. Springer, 2009.

[26] Ivan Damgård, Valerio Pastro, Nigel Smart, and Sarah Zakarias. Multiparty computation from somewhat homomorphic encryption. In *Annual Cryptology Conference*, pages 643–662. Springer, 2012.

[27] Daniel Demmler, Thomas Schneider, and Michael Zohner. Aby-a framework for efficient mixed-protocol secure two-party computation. In *NDSS*, 2015.

[28] Hongmei Deng, Anindo Mukherjee, and Dharma P Agrawal. Threshold and identity-based key management and authentication for wireless ad hoc networks. In *Information Technology: Coding and Computing, 2004. Proceedings. ITCC 2004.*, pages 107–111, 2004.

[29] Marten van Dijk, Craig Gentry, Shai Halevi, and Vinod Vaikuntanathan. Fully homomorphic encryption over the integers. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques, EUROCRYPT 2010*, pages 24–43, 2010.

[30] Jack Doerner, David Evans, and Abhi Shelat. Secure stable matching at scale. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 1602–1613, 2016.

[31] Mahir Can Doganay, Thomas B Pedersen, Yücel Saygin, Erkay Savaş, and Albert Levi. Distributed privacy preserving k-means clustering with additive secret sharing. In *the 2008 international workshop on Privacy and anonymity in information society*, pages 3–11. ACM, 2008.

[32] Cynthia Dwork and Aaron Roth. The algorithmic foundations of differential privacy. *Foundations and Trends in Theoretical Computer Science*, 9(3–4):211–407, 2014.

[33] Ariel Ekblaw, Asaph Azaria, John D Halamka, and Andrew Lippman. A case study for blockchain in healthcare:“medrec” prototype for electronic health records and medical research data. In *Proceedings of IEEE open & big data conference*, page 13, 2016.

[34] Yousef Elmehdwi, Bharath K Samanthula, and Wei Jiang. Secure k-nearest neighbor query over encrypted data in outsourced environments. In *Data Engineering (ICDE), 2014 IEEE 30th International Conference on*, pages 664–675. IEEE, 2014.

[35] Uriel Feige, Amos Fiat, and Adi Shamir. Zero-knowledge proofs of identity. *Journal of Cryptology*, 1(2):77–94, 1988.

[36] Atsushi Fujioka, Tatsuaki Okamoto, and Kazuo Ohta. A practical secret voting scheme for large scale elections. *Proc Auscrypt92 Gold Coast Queensland Australia Dec*, 718:244–251, 1992.

[37] Adrià Gascón, Phillipp Schoppmann, Borja Balle, Mariana Raykova, Jack Doerner, Samee Zahur, and David Evans. Privacy-preserving distributed linear regression on high-dimensional data. *Cryptology ePrint Archive*, 2016.

[38] Craig Gentry, Shai Halevi, Hugo Krawczyk, Bernardo Magri, Jesper Buus Nielsen, Tal Rabin, and Sophia Yakoubov. Yoso: You only speak once: Secure mpc with stateless ephemeral roles. In *Annual International Cryptology Conference*, pages 64–93. Springer, 2021.

[39] Oded Goldreich. Secure multi-party computation. *Manuscript. Preliminary version*, 78, 1998.

[40] Oded Goldreich, Silvio Micali, and Avi Wigderson. Proofs that yield nothing but their validity or all languages in np have zero-knowledge proof systems. *Journal of the ACM (JACM)*, 38(3):690–728, 1991.

[41] Lorenzo Grassi, Christian Rechberger, Dragos Rotaru, Peter Scholl, and Nigel P Smart. Mpc-friendly symmetric key primitives. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 430–443, 2016.

[42] Shay Gueron, Yehuda Lindell, Ariel Nof, and Benny Pinkas. Fast garbling of circuits under standard assumptions. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, pages 567–578, 2015.

[43] Danny Harnik, Yuval Ishai, Eyal Kushilevitz, and Jesper Buus Nielsen. Ot-combiners via secure computation. In *Theory of Cryptography Conference*, pages 393–411. Springer, 2008.

[44] Thomas Icart. How to hash into elliptic curves. In *Advances in Cryptology-CRYPTO 2009*, pages 303–316. 2009.

[45] Geetha Jagannathan and Rebecca N. Wright. Privacy-preserving distributed k-means clustering over arbitrarily partitioned data. In *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 593–599, 2005.

[46] Somesh Jha, Luis Kruger, and Patrick Mcdaniel. Privacy preserving clustering. In *European Symposium on Research in Computer Security, ESORICS 2005*, pages 397–417, 2005.

[47] Liina Kamm and Jan Willemson. Secure floating point arithmetic and private satellite collision analysis. *International Journal of Information Security*, 14(6):531–548, 2015.

[48] Jonathan Katz, Steven Myers, and Rafail Ostrovsky. Cryptographic counters and applications to electronic voting. In *International Conference on the Theory and Applications of Cryptographic Techniques*, pages 78–92, 2001.

[49] Marcel Keller. Mp-spdz: A versatile framework for multi-party computation. In *Proceedings of the 2020 ACM SIGSAC conference on computer and communications security*, pages 1575–1590, 2020.

[50] Marcel Keller, Peter Scholl, and Nigel P Smart. An architecture for practical actively secure mpc with dishonest majority. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, pages 549–560, 2013.

[51] Aggelos Kiayias and Moti Yung. Self-tallying elections and perfect ballot secrecy. *Lecture Notes in Computer Science*, 2274:141–158, 2002.

[52] Meeser F L. Decentralized, transparent, trustless voting on the ethereum blockchain, 2017.

[53] Andrei Lapets, Frederick Jansen, Kinan Dak Albab, Rawane Issa, Lucy Qin, Mayank Varia, and Azer Bestavros. Accessible privacy-preserving web-based data analysis for assessing and addressing economic inequalities. In *Proceedings of the 1st ACM SIGCAS Conference on Computing and Sustainable Societies*, pages 1–5, 2018.

[54] Jin Li, Lichao Sun, Qiben Yan, Zhiqiang Li, Witawas Srisa-an, and Heng Ye. Significant permission identification for machine learning based android malware detection. *IEEE Transactions on Industrial Informatics*, 2018.

[55] Ping Li, Jin Li, Zhengan Huang, Chong Zhi Gao, Wen Bin Chen, and Kai Chen. Privacy-preserving outsourced classification in cloud computing. *Cluster Computing*, 21(1):1–10, 2017.

[56] Ping Li, Jin Li, Zhengan Huang, Tong Li, Chong Zhi Gao, Siu Ming Yiu, and Kai Chen. Multi-key privacy-preserving deep learning in cloud computing. *Future Generation Computer Systems*, 74(C):76–85, 2017.

[57] Tong Li, Zhengan Huang, Ping Li, Zheli Liu, and Chunfu Jia. Outsourced privacy-preserving classification service over encrypted data. *Journal of Network and Computer Applications*, 106:100–110, 2018.

[58] Tong Li, Jin Li, Zheli Liu, Ping Li, and Chunfu Jia. Differentially private naive bayes learning over multiple data sources. *Information Sciences*, 444:89–104, 2018.

[59] Ye Li, Zoe L. Jiang, Lin Yao, Xuan Wang, S. M. Yiu, and Zhengan Huang. Outsourced privacy-preserving c4.5 decision tree algorithm over horizontally and vertically partitioned dataset among multiple parties. *Cluster Computing*, (2):1–13, 2017.

[60] Yehuda Lindell and Benny Pinkas. Privacy preserving data mining. In *Annual International Cryptology Conference, CRYPTO 2000*, pages 36–54, 2000.

[61] Bin Liu, Yurong Jiang, Fei Sha, and Ramesh Govindan. Cloud-enabled privacy-preserving collaborative learning for mobile sensing. In *Proceedings of the 10th ACM Conference on Embedded Network Sensor Systems*, pages 57–70, 2012.

[62] Dongxi Liu, Elisa Bertino, and Xun Yi. Privacy of outsourced k-means clustering. In *the 9th ACM symposium on Information, computer and communications security*, pages 123–134. ACM, 2014.

[63] Joseph K Liu, Victor K Wei, and Duncan S Wong. Linkable spontaneous anonymous group signature for ad hoc groups. In *Australasian Conference on Information Security and Privacy*, pages 325–335, 2004.

[64] Joseph K Liu and Duncan S Wong. Linkable ring signatures: Security models and new schemes. In *International Conference on Computational Science and Its Applications*, pages 614–623, 2005.

[65] Xiaoyan Liu, Zoe L Jiang, Siu-Ming Yiu, Xuan Wang, Chuting Tan, Ye Li, Zechao Liu, Yabin Jin, and Junbin Fang. Outsourcing two-party privacy preserving k-means clustering protocol in wireless sensor networks. In *Mobile Ad-hoc and Sensor Networks (MSN), 2015 11th International Conference on*, pages 124–133. IEEE, 2015.

[66] Patrick McCorry, Siamak F Shahandashti, and Feng Hao. A smart contract for boardroom voting with maximum voter privacy. In *International Conference on Financial Cryptography and Data Security*, pages 357–375, 2017.

[67] Payman Mohassel, Mike Rosulek, and Ni Trieu. Practical privacy-preserving k-means clustering. *Proceedings on privacy enhancing technologies*, 2020.

[68] Daniel Morales, Isaac Agudo, and Javier Lopez. Private set intersection: A systematic literature review. *Computer Science Review*, 49:100567, 2023.

[69] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. 2008.

[70] B Ortutay. Ibm to invest -billion in new 'internet of things' unit. *Globe and Mail*, 2015.

[71] Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *International Conference on the Theory and Applications of Cryptographic Techniques*, pages 223–238. Springer, 1999.

[72] Sankita Patel, Viren Patel, and Devesh Jinwala. Privacy preserving distributed k-means clustering in malicious model using zero knowledge proof. In *International Conference on Distributed Computing and Internet Technology*, pages 420–431. Springer, 2013.

[73] Sankita Patel, Mitali Sonar, and Devesh C Jinwala. Privacy preserving distributed k-means clustering in malicious model using verifiable secret sharing scheme. *International Journal of Distributed Systems and Technologies (IJDST)*, 5(2):44–70, 2014.

[74] Rahul Rachuri and Peter Scholl. Le mans: Dynamic and fluid mpc for dishonest majority. In *Annual International Cryptology Conference*, pages 719–749. Springer, 2022.

[75] Fang-Yu Rao, Bharath K Samanthula, Elisa Bertino, Xun Yi, and Dongxi Liu. Privacy-preserving and outsourced multi-user k-means clustering. In *Collaboration and Internet Computing (CIC), 2015 IEEE Conference on*, pages 80–89. IEEE, 2015.

[76] Peter Rindal and Mike Rosulek. Faster malicious 2-party secure computation with {Online/Offline} dual execution. In *25th USENIX Security Symposium (USENIX Security 16)*, pages 297–314, 2016.

[77] Dragos Rotaru, Nigel P Smart, and Martijn Stam. Modes of operation suitable for computing on encrypted data. *IACR Transactions on Symmetric Cryptology*, pages 294–324, 2017.

[78] Jun Sakuma and Shigenobu Kobayashi. Large-scale k-means clustering with user-centric privacy-preservation. *Knowledge and Information Systems*, 25(2):253–279, 2010.

[79] Berry Schoenmakers. Mpyc—python package for secure multiparty computation. In *Workshop on the Theory and Practice of MPC. https://github. com/lschoe/mpyc*, 2018.

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

[81] Seung Won Shin, Phillip Porras, Vinod Yegneswara, Martin Fong, Guofei Gu, and Mabry Tyson. Fresco: Modular composable security services for software-defined networks. In *20th annual network & distributed system security symposium*. Ndss, 2013.

[82] Yonatan Sompolinsky and Aviv Zohar. Secure high-rate transaction processing in bitcoin. In *International Conference on Financial Cryptography and Data Security*, pages 507–527, 2015.

[83] Ebrahim M Songhori, Siam U Hussain, Ahmad-Reza Sadeghi, Thomas Schneider, and Farinaz Koushanfar. Tinygarble: Highly compressed and scalable sequential garbled circuits. In *2015 IEEE Symposium on Security and Privacy*, pages 411–428. IEEE, 2015.

[84] Maneesh Upmanyu, Anoop M Namboodiri, Kannan Srinathan, and CV Jawahar. Efficient privacy preserving k-means clustering. In *Pacific-Asia Workshop on Intelligence and Security Informatics*, pages 154–166. Springer, 2010.

[85] Jaideep Vaidya and Chris Clifton. Privacy-preserving k-means clustering over vertically partitioned data. In *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 206–215, 2003.

[86] Xiao Wang, Samuel Ranellucci, and Jonathan Katz. Authenticated garbling and efficient maliciously secure two-party computation. In *Proceedings of the 2017 ACM SIGSAC conference on computer and communications security*, pages 21–37, 2017.

[87] Gavin Wood. Ethereum: A secure decentralised generalised transaction ledger. *Ethereum project yellow paper*, 151:1–32, 2014.

[88] Lei Xu, Chunxiao Jiang, Jian Wang, Jian Yuan, and Yong Ren. Information security in big data: privacy and data mining. *IEEE Access*, 2:1149–1176, 2014.

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

[90] Bin Yu, Joseph K. Liu, Amin Sakzad, Surya Nepal, Ron Steinfeld, Paul Rimba, and Man Ho Au. Platform-independent secure blockchain-based voting system. In *Information Security: 21st International Conference,*, page 369, 2018.

[91] Samee Zahur and David Evans. Obliv-c: A language for extensible data-oblivious computation. *Cryptology ePrint Archive*, 2015.

[92] Justin Zhan. Privacy-preserving collaborative data mining. *IEEE Computational Intelligence Magazine*, 3(2):31–41, 2008.

[93] Zhichao Zhao and T.-H. Hubert Chan. How to vote privately using bitcoin. In *Information and Communications Security*, pages 82–96, 2016.

[94] Chaoshun Zuo, Zhiqiang Lin, and Yinqian Zhang. Why does your data leak? uncovering the data leakage in cloud from mobile apps. In *2019 IEEE Symposium on Security and Privacy (SP)*, pages 1296–1310. IEEE, 2019.

[95] Vinod Vaikuntanathan Zvika Brakerski. Efficient fully homomorphic encryption from (standard) lwe. *SIAM Journal on Computing*, 43(2):831–871, 2011.