EFFICIENT SECURE COMPARISON IN THE DISHONEST MAJORITY MODEL

A Dissertation presented to

the Faculty of the Graduate School

at the University of Missouri

In Partial Fulfillment

of the Requirements for the Degree

Doctor of Philosophy

---

by

Ali Ataeemh Al-lami

Dr. Wei Jiang, Thesis Supervisor

December, 2021

The undersigned, appointed by the Dean of the Graduate School, have examined the dissertation entitled:

EFFICIENT SECURE COMPARISON
IN THE DISHONEST MAJORITY MODEL

presented by Ali Ataeemh Al-lami,
a candidate for the degree of Doctor of Philosophy and hereby certify that, in their opinion, it is worthy of acceptance.

Dr. Wei Jiang

Dr. Jian Lin

Dr. Dan Lin

Dr. Khaza Anuarul Hoque

ACKNOWLEDGMENTS

I would like to express my sincere gratitude to Iraq's Higher Committee for Education Development for supporting my Ph.D. program. The country receives an essential contribution by supporting the research and development of young professionals in their program. In addition, I would like to thank my adviser, Dr. Wei Jiang, for his insightful remarks and recommendations on this dissertation. He has been a continual source of inspiration, advice, encouragement, and support. I feel fortunate to have such an exceptional, compassionate advisor like him, and I am grateful for everything he has done for me. My words do not seem adequate to describe his genuine contribution to this adventure.

Additionally, I am thankful for the time, expertise, and comments of all the members of the committee who provided helpful feedback and recommendations, Dr. Lin, Dr. Hoque, and Dr. Lin. I am also thankful to the University of Missouri, the College of Engineering, the EECS Department, and all the faculty and staff for their considerate guidance and help in navigating the entire process. Finally, I cannot forget to thank my family and friends for all their unconditional support.

TABLE OF CONTENTS

LIST OF TABLES

Table Page

LIST OF FIGURES

Figure Page

ABSTRACT

Secure comparison (SC) is an essential primitive in Secure Multiparty Computation (SMC) and a fundamental building block in Privacy-Preserving Data Analytics (PPDA). Although secure comparison has been studied since the introduction of SMC in the early 80s and many protocols have been proposed, there is still room for improvement, especially providing security against malicious adversaries who form the majority among the participating parties. It is not hard to develop an SC protocol secure against malicious majority based on the current state-of-the-art SPDZ framework. SPDZ is designed to work for arbitrary polynomially-bounded functionalities; it may not provide the most efficient SMC implementation for a specific task, such as SC. In this thesis, we propose a novel and efficient compiler specifically designed to convert most existing SC protocols with semi-honest security into the ones secure against the dishonest majority (malicious majority). We analyze the security of the proposed solutions using the real-ideal paradigm. Moreover, we provide computation and communication complexity analysis. Comparing to the current state-of-the-art SC protocols Rabbit and edaBits, our design offers significant performance gain. The empirical results show that the proposed solution is at least 5 and 10 times more efficient than Rabbit in run-time and communication cost respectively.

Chapter 1

Introduction

Comparison serves as one of the most fundamental operators in various data analytics. When the data considered under these applications contain sensitive information and are from multiple sources, privacy-preserving protocols may have to be adopted to protect the data and the outcomes. Secure Multiparty Computation (SMC) primitives are essential building blocks for developing many existing privacy-preserving protocols.

The first example that outlines the secure comparison protocol is the known "millionaire" problem which is proposed by Yao in 1982 [1]. In this problem, a group of millionaires would like to figure out which person between them is the most prosperous. However, being millionaires, they do not want to reveal accurately how wealthy they are. At the first glance, the problem might sound unsolvable due to the fact that the input (the wealth) is required to be kept secure. However, this complication can be determined by either finding a trusted third party who can do the computation on behalf of the millionaires or via employing secure multiparty computation. SMC plays the same role as the trusted third party as it guarantees that the distrustful parties (the millionaires) learn nothing about each other input besides the output (determine which person is the richest).

Figure 1.1 shows a real case scenario of SMC in which the patients records belong to multiple health organizations. The health organizations outsource their data to cloud servers. Correlating this information can provide major benefits to the patients as well as the health organizations. In this case, an authorized user (physician) wishes to query the patients records. The issue here is that patient records have sensitive information. Considering the data comes from different sources, privacy-preserving data analytics (PPDA) protocols may have to be adopted to protect the data and the outcomes. SMC primitives are the building blocks for many existing PPDA protocols. Secure comparison is the core operation in many of these primitives.

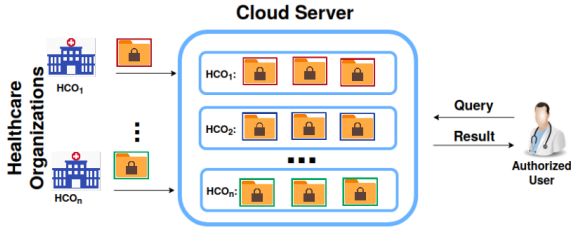

Figure 1.1: Real case scenario of SMC

Although SMC techniques provide a solid guarantee of personal privacy and data security, they are computationally expensive. As a result, notable efforts have been devoted to developing efficient SMC primitives for the last three decades, including secure comparison (SC). Current SC implementations can be classified into several categories based on the underlying building blocks, such as garbed circuits [2, 3], homomorphic encryption [4,5], secret sharing [6–9], and the SPDZ framework [10–13]. Additionally, the existing SC protocols can also be classified based on their security guarantees [14–16] as follows:

- Semi-honest: If a protocol is secure under the semi-honest assumption, the participating parties expect to follow the execution requirement of the protocol. However, they may use what they see during the execution to compute more than they need to know.

- Malicious: If a protocol is secure under the malicious assumption, the participating parties can deviate arbitrarily from the normal execution of the protocol. The adversary under this model can cheat the system with a negligible probability.

- Covert: The covert model [15, 16], a sub-class of the malicious model, takes advantage of both semi-honest and malicious models. It allows the participating parties to diverge arbitrarily (as with the malicious) but provides certain detectability guarantees of such behaviors. As a result, protocols that satisfy covert security are more efficient than those that guarantee full malicious security.

In general, the semi-honest adversarial model often leads to more efficient privacy-preserving protocols than the malicious model, but the malicious model is less restrictive and thus more realistic.

The adversary may corrupt a protocol based on different strategies. The corruption strategy answers the subject of when and how parties are corrupted. There are three main models as follows:

1. Static corruption model: In this model, t parties are controlled by an adversary. These parties are fixed before the protocol begins. Honest parties remain honest, and corrupted parties remain corrupted until the end of protocol execution.

2. Adaptive corruption model: Rather than holding a fixed set of corrupted parties, adaptive adversaries have the ability to corrupt the parties during the computation. The choice of whom to corrupt and when can be arbitrarily decided by the adversary. It may depend on its view of the execution hence the name adaptive. This strategy models the threat of a party that is honest initially and next changes its behavior. It is worth noting that once a party is corrupted, it remains corrupted from that point on in this model.

3. Proactive security model [17, 18]: This model takes into account the likelihood that parties are corrupted for a limited period of time. Therefore, honest parties may become corrupted during the protocol calculations, similar to the adaptive adversarial model, but dishonest parties may also become honest. The proactive model occurs when the threat is an external adversary who may break into the system while a secure computation continues. In some scenarios, when breaches are discovered, the systems can be cleaned. Therefore, it makes the adversary loses control of some devices, leading the parties to be honest again. The security guarantee under this model is that the adversary can only learn what is determined from the local state of a machine it was corrupted. This sort of adversary is sometimes called a mobile adversary.

The motivation for working on secure comparison protocols in the SMC paradigm is that the core operations and models in the multiparty computation are now well established. Finding more efficient methods for complicated operations and building blocks, on the other hand, remains an open research challenge. Thus, despite the significant development of the SMC and the secure comparison operation, there is still much room for improvement especially for the case of dishonest majority where the adversary controls the majority of the parties. As a result, it is an exciting field to work in. The research described in this thesis concentrates on the efficient implementation of secure comparison operations. Especially, the research findings stress on the "less than" operation. Progress on this operation will hopefully be used to attain results with more complicated protocols in the future.

1.1 SMC Applications

During the last three decades, SMC shows many great theoretical examples that can be valuable. The utilization varies from enabling private DNA comparisons for medi-

cal and other purposes to gathering statistics without revealing the aggregate results and many more. Due to the rapid development in the communication infrastructure and the SMC techniques, SMC is implemented in various real-world use cases, and usage is growing fast. In this section, we provide several examples of SMC applications that have been deployed in this area.

1. Boston wage gap [19]: The Boston Women's Workforce Council adopted SMC in 2017 to calculate compensation statistics of more than 166K employees across 114 companies, including around 16 of the Greater Boston area workforce. Since companies are not willing to provide their plain data due to privacy concerns, the application of SMC was essential. The findings imply that the gender disparity in the Boston area is even wider than the US Bureau of Labor Statistics previously anticipated. This is an important example proving that SMC can be used for social advantage.

2. Advertising conversion [20]: Conversion rates from ads to real purchases requires to be calculated accurately. Google computes the intersection's size between the list of people showing an advertisement and people purchasing the advertised goods. The purchase link to the displayed advertisement cannot be tracked if the items are not purchased online. Google and the firm paying for the advertisement must disclose their respective lists to determine the intersection size. Google utilizes a protocol for the privacy-preserving set intersection without revealing anything, but the size of the intersection [20]. Although it is not the most efficient protocol available today, it is straightforward and fits the company's computing needs.

3. SMC for cryptographic key protection [21]: To protect cryptographic keys, some firms are utilizing threshold cryptography as an alternative to legacy hardware. Threshold cryptography allows to perform cryptographic operations (such as

decryption and signing) without storing the private key anywhere. SMC is not used in this application to transfer private information between parties. Rather, SMC is used by a single organization to create keys and perform cryptographic operations without the key ever being stored somewhere where it may be stolen. By dispersing the key shares over multiple settings, an attacker will have difficulty stealing all of them and obtaining the key. SMC may also be used to preserve the signing keys used to protect cryptocurrencies and other digital assets. The cryptographic enforcement of tight standards for allowing financial transactions or sharing keys between custody providers and clients is enabled by specifying general quorums.

4. Government collaboration [22]: Many government entities hold information on residents, and correlating that information can provide major benefits. However, the privacy issues associated with private data pooling may deter governments from doing so. For example, in 2000, Canada canceled a scheme to pool citizen data in response to accusations that it was creating a "big database." Estonia gathered encrypted income tax information and higher education records using SMC to determine if students who work while studying are more likely to fail than those who do not focus entirely on their academics. Using SMC, the government was certain that all data security and tax secrecy requirements would be fulfilled without sacrificing data utility.

5. Privacy-preserving analytics [23]: Machine learning is becoming increasingly popular in a variety of fields. SMC is capable of running machine learning models on data without disclosing the model (which contains valuable intellectual property) to the data owner or the data to the model owner. Furthermore, statistical studies can be performed across companies for anti-money laundering, risk core calculations, and other purposes.

6. Secure statistical analysis of income tax records [24, 25]: Bogdanov et al. describe using SMC for the Estonian government to accomplish a secure statistical analysis of income tax records. The latter work analyzed an extensive database with over 600K students and 10M tax records. Again, simple statistics were used, and the work heavily uses the fact that secure additions are non-interactive.

7. Private benchmarking applications [26]: SMC is used as a private benchmarking application, allowing institutions to jointly analyze client risks while protecting customer data privacy. Secure linear programming is employed using SMC, which is a deeply hard operation that requires either extremely large integer arithmetic (to simulate real numbers without overflow) or secure floating-point arithmetic. It is worth noting that using Boolean circuits would be impracticable in both situations.

8. Satellites collision prevention: SMC has been presented as a technique for preventing satellite collisions by safely integrating collision detection with sensitive position and trajectory data. Kamm et al. [27] describe how to implement the required conjunction analysis algorithms in SMC using a secret-sharing system. It should be noted that this application is based on secure floating-point operations.

1.2 Thesis Contribution

This thesis proposes an efficient secure comparison protocol and a generic compiler to transfer any semi-honest secure comparison protocol to a secure comparison protocol under dishonest majority (majority malicious) setting. Below we highlight some more details of our contribution.

1. Design an efficient comparison protocol (SC_{3P}) that is tailored to the honest majority setting. The protocol works under three parties, one of them is designated to facilitate the secret sharing secure computation. As a result, the protocol does not require secure multiplication which represents the overhead of secure comparison operation. Moreover, we used domain reduction method which helps in reducing the message complexity from quadratic to linear in terms of the bit length.

2. Generic compiler for secure comparison: we propose a novel technique, termed as *randomized replication*, to develop a generic compiler that transforms any semi-honestly secure comparison protocol to be secure against dishonest majority (malicious majority). The proposed compiler is generic such that a newly developed and more efficient secure comparison protocol can be used without changing the rest of the code or the structure of that protocol. The compiler overcomes the high cost of the SPDZ protocols using a simple yet effective treatments as follows:

   (a) We present an efficient compiler by removing the ZK-Proof which represents the bottleneck of the SPDZ compiler. Our compiler only depends on executing the protocol κ times. Where κ is the soundness security parameter.

   (b) The soundness parameter κ is independent of the underling HE. This leads to a small polynomial dimension N; thus, a short ciphertext.

   (c) The compiler achieves security in both covert and malicious models only by adjusting the κ parameter. It is more efficient than the existing solutions for covert security based on the cut-and-choose technique. By randomly permuting the input ordering, the error probability decreases exponentially as the number of copies increase.

(d) Achieving higher security comparing to the state of the art Rabbit protocol. Since Rabbit utilizes edabit [28] to generate random unknown values associated with its random unknown bits. Edabit requires to check the consistency of the random values. Therefore, it utilizes cut-and-choose technique which requires to set a statistical security paramter. Edabit is implemented in MP-SPDZ [29] with only 40-bit statistical security. This leads to reduce Rabbit security to only 40-bit. It is worth noting that our compiler has no limit on the statistical security parameter.

3. We provide a formal security analysis using the real-ideal paradigm. In addition to that, we give theoretical computation and communication analysis for the proposed solutions. Moreover, we implement our compiler using SEAL library [30] and c++. We run extensive experiments and the empirical results show that the proposed solution is at least 5 and 10 times more efficient than Rabbit in run-time and communication cost respectively.

1.3 Overview

In this section, we cover the problem addressed in this work. Initially we define the problem and then describe the adversary model for the proposed protocols.

1.3.1 Problem Definition

This work focuses on secure comparison protocols. More specifically, we consider the client-server computing model where clients outsource their data and analytics tasks to two or more independent servers. Most existing SMC solutions are applicable in the model. Since the clients are not involved in protocol execution, the servers are commonly referred to as the participating parties. As a result, given a and b are non-

negative integers secretly shared among n parties: $P_1, \ldots, P_n$. Let $[a]$ and $[b]$ be the secret shares of a and b. Both values are bounded by $\lfloor \frac{p}{2} \rfloor$, p is a prime of $l = \lceil \log_2 p \rceil$ bits. Our protocol implement the comparison functionality as follows:

$$f(a,b) = \begin{cases} 0 & \text{if } a \geq b \\ 1 & \text{if } a < b \end{cases} \tag{1.1}$$

We define the secure comparison protocol as follows

$$\text{SC}(\langle P_i, [a]^{P_i}, [b]^{P_i} \rangle) \rightarrow \langle P_i, [\delta]^{P_i} \rangle \tag{1.2}$$

1.3.2 Security Guarantee and Threat Model

Let n denote the number of parties/servers, and up to $n-1$ parties can be malicious. The assumption of computing power of these parties depends on the actual design and implementation of the SC protocols being transformed using the proposed compiler. For example, if an SC protocol assumes the parties are computationally unbounded, then the same assumption holds under our compiler. The security of an SMC protocol has several common and essential criteria:

- Privacy: the private input data of an honest party is not disclosed to the other parties during protocol execution.
- Correctness: in presence of malicious behaviors, the honest parties can still receive the correct output.
- Fairness: either every party receives the correct output or no parties receive the correct output.
- Detectability: any malicious behaviors can be detected.

Any SMC protocols have to guarantee privacy, but the other properties may or may

not be achieved depending on the number of malicious parties. For example, robustness may be achievable if the number of malicious parties is less than $\frac{n}{3}$ with Shamir's secret sharing scheme.

Under the malicious majority setting, in addition to privacy, the existing SMC protocols only guarantee detectability of malicious behaviors. Malicious behaviors generally mean collusion among the parties and not following the protocol, all of which is equivalent to changing or modifying the shares. For instance, suppose a value $v = v_1 + v_2 + v_3 \bmod p$ is secretly shared among three parties: P_1, P_2 and P_3. Each P_i has the share v_i. Suppose v is a private input of an SMC protocol, and P_1 and P_2 are malicious. Then any malicious behaviors of P_1 and P_2 are equivalent to using v_1' and v_2' as their shares during protocol execution where v_1' and v_2' may or may not be the same as v_1 and v_2. Therefore, except for prematurely aborting the protocol, detecting malicious behaviors actually means the protocol can detect or verify if the shares have been modified. To summary, our proposed compiler provides detectability when the number of colluding and malicious parties is bounded by $n - 1$, and the participating parties have either limited or unlimited computing power depending on the implementation of the underlying SC protocols.

1.4 Thesis Structure

This chapter introduces the secure comparison problem in the multiparty computation setup (SMC) and the motivation for working with multiparty computation. Chapter 2 provides a comprehensive review of the related works as well as the necessary notations and conventions. Chapter 3 presents the proposed efficient, secure comparison protocols with correctness, security, and complexity analysis in addition to the evaluation results. In chapter 4, we show our treatment to convert semi-honest secure comparison protocols to malicious security under a (dishonest majority) major-

ity malicious setup. The chapter includes correctness, security as well as complexity analysis along with the performance results. Finally, chapter 5 concludes the work and provides future works.

Chapter 2

Related Work and Background

A large and interesting work has been proposed in the area of secure comparison (SC), in general, as well as within the field of Privacy-preserving Data Analysis in the SMC. Secure comparison is a highly utilized functionality in many applications; due to this, there is a high demand for efficient SC solutions. In this chapter, we begin by covering the related work in the SMC as well as the background details of concepts used within this work.

This chapter first covers comprehensive works in SMC using the the malicious minority model (also known as the honest majority model) and the malicious majority model (also known as the dishonest majority model) . Aside from the semi-honest model, these two models define the framework in which secure comparison protocols can be implemented. Then we give a detailed list of studies in the field of secure comparison. Because we use fully homomorphic encryption FHE in our design, we give a list of relevant work in this area for completeness. We also provide a list of work related to the oblivious transfer (OT) protocols which is used to facilitate the secure multiplication in the presence of a malicious adversary. Finally, we present preliminaries and notations used throughout this thesis.

2.1 Related Work

2.1.1 SMC Against Honest Majority

A number of SMC solutions have been proposed to address the security issues under the majority honest malicious adversary model. Here we summarize a few representative work in this area. A verifiable secret sharing scheme is proposed in [31] which uses Shamir's secret sharing and homomorphic commitments based on the discrete log assumption. The commitments are updated along the computations, and any malicious changes to the computation will lead to inconsistent commitments. it is worth noting that this commitment scheme is computationally expensive. Combining dispute control [32] and utilizing a designed party for intermediate computation [33], more efficient solutions is introduced in [34]. It has linear complexity based on the circuit size, and its verification technique has error probability negligible in terms of pre-defined security parameter. To remove this error probability, [35] uses hyper-invertible matrices to perform batched correctness check of shares that leads to a perfectly secure solution. In [36], a technique of 4-consistent tuples of shares is proposed to improve the communication complexity given in [35] by removing the quadratic terms of the multiplicative depth of the circuit. To ensure perfect security in presence of malicious parties, all these solutions assume the number of malicious parties is less than $\frac{n}{3}$.

The work in [37] suggests a solution with linear complexity and less than $\frac{n}{2}$ parties can be malicious which is built based on the solution given in [38]. In [39], a framework was introduced to allow computations performed by using a semi-honest protocol along with verification steps to detect malicious behaviors with a very high probability. It presents an efficient way to verify the correctness of a set of Beaver triples [40]. In [41], a circuit randomization technique was proposed to verify the consistencies between two executions: one on the original circuit and one on a randomized circuit by multiplying the inputs with a random value.

Both works [39,41] assume that the multiplication protocol is secure up to additive attack [42, 43],[1] and the number of malicious parties is less than $\frac{n}{2}$.

2.1.2 SMC Against Dishonest Majority

When the number of malicious parties become the majority (up to $n-1$), designing efficient SMC protocol gets more and more challenging. The well-known SPDZ framework utilizes information theoretic MACs on top of additive shared secrets over a finite filed $\mathbb{F}_p$ to guarantee privacy and correctness. The process carries on two phases as follows:

- The offline phase prepares multiplication triples also known as Beaver's triples [44]. Since these triples are independent; they can be generates as batches in parallel mode for efficiency.
- The online phase consumes the triples that are generated in the offline phase to perform the actual multiplication calculation.

SPDZ protocol relies on BDOZ [45] protocol in which the offline phase relies on pairwise multiplication carries on linearly homomorphic encryption. Zero-knowledge proof ZKP utilizes in the offline phase to ensure dishonest parties cannot deviate from the protocol. It requires a total of $O(n^2)$ ZK-Proof per multiplication triple of BDOZ. Pairwise MACs applies to authenticate a secret sharing between n-parties.

Damgard et al. [10] upgrade the underling linearly homomorphic encryption used by BDOZ to a somewhat homomorphic encryption (SHE) based on BV scheme [46]. This allows to decrease the number of ZK-Proof per multiplication triple by a factor of n. Moreover, the pairwise MACs is replaced by a global MAC which allows to authenticate the secret value itself instead of the shares which previously is quadratic in n.

[1] Additive attack means that an adversary can add a value to the output of a multiplication.

In [11], Damgard et al. provide two new techniques based on cut-and-choose. The first one produces covert security offering high efficiency. The second method guarantee actively secure protocol with ZK-Poof asymptotically more efficient than those utilized in [10]. However, this method requires a high memory which makes it hard to be implemented.

The inefficiency of an actively secure offline phase that relies on the SHE comes from the fact that SHE requires an expensive zero knowledge proofs or other cut and choose techniques. To overcome this limitation, a new family of protocols offer security in the presence of $n-1$ dishonest parties which utilizes Oblivious Transfer OT. Nielsen et al. [47] offer a two-party protocol for binary circuits relying on OT extensions known as TinyOT. It has a throughput of roughly 10k in the field $\mathbb{F}_2$ triples per second. To overcome any leakage on the secret correlation, they rely on consistency checks and privacy amplification techniques with an overhead of roughly 7.3 of the base OT calls. To remove the leakage on a and b during the process of generating the triples, they rely on a combining procedure. The combining procedure requires to be done twice, one to remove the leakage on a and the other one is to eliminate the leakage on b. It works similarly to the sacrifice technique by batching triples randomly into buckets such that if one of them is secure the resulting triples stay secure.

Larraia et al. [48] show how to extend TinyOT to the multi-party setting and accommodate it to match the online phase of the SPDZ protocols. Triple sacrifice lies in the hart of TinyOT to guarantee the triple correctness. To remove the possibility of any leakage from the triples, a combining technique is utilized. However, in a small field simple pairwise checks may not suffice. Therefore, bucketing technique is used which is an expensive method with an overhead of around 3-8 times per check, based on the statistical security parameter as well as the number of triples.

It is worth noting that in all the previously mentioned papers the focus is on

improving the offline phase. While the online phase is identical across all of them. This is where the MiniMAC [49] comes in the picture in order to speed up the online phase. The idea of MiniMAC is to reduce the size of the MACs in the online phase for both the binary circuits and the arithmetic circuits over a small filed. Previously, the MAC of secret shared values has to be at least as big as the statistical security parameter κ. Therefore, the field should offer enough room to cover κ. MiniMAC allows to combine a vector of bits at once into a codeword, offering a constant MAC size. The implementation of the MiniMAC technique first appears in [50] which shows that MiniMAC offers faster performance than TinyOT by accomplishing many operations in parallel. It is worth mentioning that the first dedicated offline phase for the MiniMAC is proposed in [51, 52].

Frederiksen et al. [51, 52] shows how to construct the offline phase based on oblivious transfer extension. Their main focus is on finite fields of characteristic two. The authors shifted their focus of the traditional view of the sender and the receiver of the OT to the use of a linear algebra method with matrices, vectors and tensor products. During the triple generation in $\mathbb{F}_{2^\kappa}$, the consistency check is dropped allowing the adversary to introduce error. The introduced error is amplified using privacy amplification step. Therefore, the sacrifice check which is performed later suffices to detect such error. This way allows to reduce the overhead of the correlated OT protocol to only 3 times of a basic OT extension. For $\mathbb{F}_2$ triple generation, the authors show that one combining procedure is enough to remove the leakage on a and b rather than two combining procedures.

MASCOT [53] is another general purpose compiler that shifts from somewhat homomorphic encryption offline phase to Oblivious Transfer based offline phase. Their main focus is on arithmetic circuit of the finite field of a prime $\mathbb{F}_p$ as well as the power of two $\mathbb{F}_{2^\kappa}$. Similar to previous works, MASCOT utilizes privacy amplification method. However, MASCOT protocol needs to eliminate leakage on only one of the

three triple values. This is done efficiently by combining correlated triples of a τ constant sized vector where $\tau \geq 3$. Combining is done using simple inner products to ensure that any leaking bits are randomly combined with non leaking bits of a public random vector. This works due to the fact that dot product sufficiently defines a universal hash function. This allows them to use the left hash lemma to guarantee a uniformly random triples with a given large enough τ. MASCOT results show that their protocol is 200 times faster than [11]. Their results emphasis that even though it requires more communication compared to the covertly secure protocol [11]. It is still 20 times faster. It is worth mentioning that the building block of [51–53] is the oblivious product evaluation based on Gilboa's method [54].

Keller et al. [12] focus on improving the offline phase. Their solution comes in two flavors. The LowGear is suited for small number of parties where the ZKP requires to be carried such that every party needs to verify every other party's proof. The highGear is designed to overcome the limitations of the LowGear by summing all proofs and only checking the sum. HighGear improves the computation cost by a factor of n. Nevertheless, the communication cost has not improved. Due to the fact that every party requires to send every proof to other parties and then sum all the received proofs.

Due to memory and bandwidth constrains, HighGear has some limitations in the security parameters. It sets the soundness security parameter Snd_sec=ZK_sec. Where Snd_sec refers to the probability that an adversary can cheat the system. While ZK_sec represents the statistical distance of the zero knowledge protocol. From a security perspective, setting a low value of Snd_sec is more reasonable than a low value of ZK_sec. Thus, setting different values is not possible in the HighGear. The effect of treating these two parameters is rather more involved. In practice, Snd_sec drastically affects the memory and the computation overhead. On the other hand, ZK_sec has very minimal effect on the execution time.

Baum et al. [13] overcome the limitations of HighGear by introducing an improved SPDZ scheme known as TopGear. TopGear treats Snd_sec and ZK_sec separately. This allows them to guarantee a higher soundness with small amount of amortization comparing to [12]. Moreover, the new scheme has smaller memory requirements. Previously, HighGear uses a binary challenge space, however, TopGear utilizes a non-binary challenge space of $2N + 1$ where N is the ring dimension for the underline encryption scheme. This allows them to achieve an arbitrary soundness security by selecting the auxiliary ciphertext $V \geq (Snd\_sec + 2)/\log_2(2N + 1)$.

Despite the improvement in the TopGear, the offline phase of the SPDZ protocol still requires a substantially large execution time. For example, secure comparison protocols require millions of triples which may take hours to be generated. The major reason for this is the high cost required by the zero knowledge protocol.

Most SMC protocols progress the secure computations of arithmetic circuit based on a finite field. Fore example, the prime filed $\mathbb{F}_p$ where p is a prime number. SPDZ$_{2^\kappa}$ works on a more natural way of integer computations modulo 2^k. This is especially useful for implementations and applications simplification. SPDZ$_{2^\kappa}$ is the first work to present such a solution under the majority malicious model. Previously, such solutions only exist in the minority malicious model. Based on their results, they offer an efficient new scheme for information-theoretic MAC which is homomorphic modulo 2^k. This novel authentication approach performs as well as well-known conventional solutions that are homomorphic across fields $\mathbb{F}_p$. The security of the MAC based on a finite field stands for the fact that non-zero values in $\mathbb{F}$ are invertible. However, in the case of a ring $\mathbb{Z}_2k$ the dishonest party can cheat with non-negligible probability. For example, the adversary could choose $a' = a + 2^{k-1}$ with cheating probability $1/2$. SPDZ$_{2^\kappa}$ is as efficient as the MASCOT protocol [53].

2.1.3 Secure Comparison

A number of methods have been proposed for the secure comparison functionality under different adversary models and mathematical domains. These protocols start by the constant round SC protocol proposed by Damgård et al [55]. In this design, secretly shared values must first be bit decomposed among the parties involved in the computation. Alternatively, the values may exist as bitwise shares initially. This means the protocol takes as input bit decomposed shares of the private values to be compared. If this procedure is necessary, though expensive, it is potentially beneficial when other bit-wise operations may be seen as advantageous. The cost incurred in this scheme for bit decomposition may be amortized somewhat across all those sub-protocols that require it. Though there is a fairly high computational complexity and communication cost, this important result demonstrates constant rounds secure comparison is indeed possible and well within feasibility. The core idea of Damgård's comparison protocol is to locate and observe the most significant bit where the two compared numbers differ. This is done by applying prefix-Or operation which is followed by a simple subtraction between every two adjacent bits. Finally, dot product is applied with the second compared number to reflect the comparison result. It is worth noting that the prefix-Or operation is the most expensive part of Damgård's comparison protocol. In the same paper, the authors offer a constant round protocol to compute prefix-Or operation by applying the symmetric boolean function principle. prefix-Or can also be computed in a logarithmic fashion as well [56].

Another technique for utilizing finite field arithmetic properties aims to influence a comparison through intermediate comparisons and some reasoning to bring the meaning of these intermediate comparisons together to generate the desired result. This strategy was pioneered by Nishide and Ohta's work [9]. Since the bit decomposition protocol is expensive, the main concept here is to carry comparison between two secretly shared values indirectly. This is done by observing that the comparison func-

tion can be determined by computing $(a < p/2), (b < p/2), (a - b \mod p < p/2)$. This leads to reduce the round complexity from 44 to only 15 round comparing to [55]. On the other hand, the communication complexity is also decreased to $279\ell + 5$ while [55] requires $205\ell + 188\ell \log_2 \ell$ where ℓ is the bit length of a prime field. It is worth noting that this work reduced the secure multiplication invocation complexity from $\mathcal{O}(\ell \log \ell)$ to $\mathcal{O}(\ell)$ comparing to the previous work.

Damgård et al. [5] propose a secure comparison protocol for integer values by utilizing additive secret sharing and homomorphic encryption. The idea of utilizing the homomorphic encryption presents by Blake et al. [57]. However, it requires a plaintext space of size exponential in ℓ. In this work, the authors proposed a novel method which allows the computation to be carried on a smaller plaintext space. This translates to smaller exponents when taking exponentiation, hence it improves the protocol efficiency. On the other hand, some of the computations can be carried using additive secret sharing rather than the homomorphic encryption which leads to further improvement.

Other improvements have since been developed that reduce the complexity and number of intermediate calculations required depending on various constraints on the domain of values that are exchanged and compared. Reistad et al. suggested an unconditionally secure comparison technique against active/adaptive attackers [58]. Furthermore, the protocol improves both the rounds and multiplication invocations complexity. However, the protocol assumes a bounded size on the compared values of less than $\lfloor \frac{p-1}{4} \rfloor$. The process starts by transforming the comparison of two secretly shared values a, b into a comparison between $2a+1, 2b$. Then, the protocol determines the comparison result by computing the less significant bit (LSB) of the difference between the later transformed values. Computing the LSB is done by randomizing and opening the difference result via adding a uniformly random value. The final result is then rectified using two xor operations.

In [59], Reistad presents a comparison protocol that is more efficient than previous constant round comparison protocols. The protocol starts by converting the comparison of two secretly shared values into a comparison between a secret value and a global known value. This step requires the parties to generate a random shared value associated with its random shared bits. Then the protocol progress on transferring the later comparison to a single shared value. The protocol then requires calculating the LSB of this value, representing the secret and global value's comparison result. Note that this value has to be smaller than $\sqrt{(}4p)$ for some prime p. One last step is required to reflect the original comparison between the two secretly shared values which takes two xor operations.

The previous protocol is improved further by Reistad et al. [60] through improving the LSB gate. The idea of extracting the LSB of bounded size presents in [61] using Paillier encrypted values when there is sufficient room in the ring. Such that $2^{\ell+s+\log n} < p$, given s is a security parameter and n represents the number of parties. To extract the LSB of a secretly shared value, the parties first generate a uniformly random unknown bit used to hide any information. Then a random mask is computed by allowing each party to input a uniformly random value of $s + \ell - 1$-bit. The later value is used to randomize the original secretly shared value and facilitate the LSB calculation using only one xor operation between a global known value and a secretly shared value. It is worth noting that this is where the comparison protocol loses its perfect security to only statistical security.

All the previously mentioned protocols work in the arithmetic circuit. In recent developments, a new family of SMC protocols allow to work in the mixed circuit of arithmetic and binary. As a result, it opens up the avenue to improve the efficiency of SC protocols further. Escudero et al. suggest a comparison protocol based on edaBits [28]. The concept is to compare two secret shared integers by extracting the MSB from a shared integer representing the difference between the two integers. If

the first integer is less than the second, the difference is negative, and the MSB equals one. The author presented a truncation procedure to obtain the MSB. It should be noted that this protocol is statistically secure. Furthermore, the statistical security requires that the shares domain be at least $p > 2^{\ell+s+1}$ for some statistical security parameter s. As a result, it introduces a large gap also known as slack between the shares and the secrete to be truncated. The comparison protocol requires two invocations of edaBits as well as a classic daBit to facilitate the conversion back from binary to arithmetic computation.

Makri et al. present a novel comparison protocol known as Rabbit [62]. Because Rabbit uses daBit and edaBit, it offers security against an active adversary in the dishonest majority setting. Furthermore, it improves the computation and communication compared to the comparison protocol present in [28]. The protocol eliminates the need for statistical security parameter in the comparison operation. This allows the domain of the shares to be smaller than that required by prior protocols, influencing the overall efficiency of the comparison protocol. Rabbit makes use of the concept of addition commutativity across rings/fields structures. More precisely expresses a sum in two ways and thereby equals the related constraint equations. The comparison relies on identifying and correcting when a sum in a specific modulus wraps around. Despite the improvement in both computation and communication, Rabbit still requires two invocations of edaBits and three invocations of Damgård less than bit comparison protocol.

2.1.4 Fully Homomorphic Encryption (FHE)

Fully Homomorphic encryption allows both addition and multiplication to be carried on the encrypted data. Due to the nature of Fully Homomorphic encryption problem, FHE schemes come with a drawback which requires to deal with the reduction of the noise before the scheme runs out of the space that allows to evaluate a function.

Therefore, FHE comes in three main types.

- Somewhat homomorphic encryption which can evaluate functions of limited complexity.
- Leveled homomorphic encryption which allows to evaluate up to L levels mainly using some modulus switching techniques.
- Fully homomorphic encryption which can evaluate an arbitrary functions by applying the concept of bootstrapping which is simply decrypting the ciphertext homomorphically to produce a fresh ciphertext with a fixed inherent noise smaller than before.

The Evolution of FHE

In the last decade, several FHE schemes were developed following Gentry's breakthrough [63, 64]. Gentry's seminal work sketches out the main theories behind the FHE. Its core is somewhat homomorphic encryption (SHE) scheme that relies on ideal lattices. Gentry shows how to transform the SHE scheme into FHE scheme by introducing the bootstrapping technique. Gentry et al [65] present the first attempt to implement Gentry's blueprint scheme, utilizing several optimizations some of which illustrated by Smart et al. [66]. The authors report a public key size of 3.2 GB while ciphertext refresh technique takes 30 minutes with security of $\lambda = 72$ bits.

Dijk et al. [67] describe a new method for constructing SHE scheme only based on elementary modular arithmetic. Their SHE uses addition and multiplication over the integer rather than working with ideal lattices over polynomial in the case of Gentry's SHE. Eventually, they apply Gentry's bootstrapping technique to convert the scheme to a fully homomorphic scheme. It is worth noting that the semantic security of this scheme relies on the well-defined search problem known as the approximate integer GCD. The problem with Dijk's scheme is the huge size of the public key which is

in $O(\lambda^{10})$. Coron et al. [68, 69] improve the previous scheme by reducing the size of public key to $O(\lambda^7)$ this is done by encrypting with a quadratic form in the public key instead of a linear form. The scheme is semantically secure based on the approximate GCD problem. In this scheme, the authors report a public key of 802 MB and a ciphertext refresh of 14 minutes. In 2013, Coron et al. [70] introduced a compression technique to reduce the public key size further from $O(\lambda^7)$ to $O(\lambda^5)$. This scheme reports public key of 10.1 MB for similar parameters of previous scheme. The scheme also presents a new modulus switching technique for DGHV scheme [67] by adapting the BGV [71] modulus switching framework.

Brakerski et al. [72] introduce a new technique for constructing FHE scheme that does not rely on the lattices assumption. The new scheme is based on the Learning with Error (LWE) assumption which is know to be at least as hard as solving hard problem in general lattices. The scheme introduces a re-linearization technique which shows how to obtain a SHE scheme. The scheme deviates from the bootstrapping general technique and relies on dimension-modulus reduction technique to convert a SHE scheme to a FHE scheme. Brakerski et al [71] improve the noise growth of the previous scheme by introducing a modulus switching technique that define a ladder q_L levels of moduli. Despite the noise improvement, the homomorphic evaluation is more complicated than before. Brakerski [73] further improves the noise reduction by introducing a scale invariant scheme, via relying on invariant perspective. The idea is to scale down by a factor of q which gives a fractional ciphertext modulo 1. In this case, the noise is not squared in the homomorphic multiplication. However, it multiplies it by a polynomial factor of $p(n)$. In [74], the authors lift the scheme in [72] from LWE to ring-LWE (RLWE). Moreover, they introduce two optimized versions of relinearization with smaller relinearization key. The problem with the scheme that depends on the LWE/RLWE is the costly multiplication that involves the relinearization step to reduce a quadratic ciphertext to a linear ciphertext. In [75], the authors

propose a new LWE technique to construct FHE scheme based on approximate eigenvector method. In this scheme, there is no need for the costly relinearization step which requires $\Omega(n^3)$ complexity. The ciphertext in this case is a matrix and the homomorphic operations is done based on matrix addition and multiplication. It is worth noting that matrix multiplication uses sub-cubic computation such as Strassen and Williams with complexity $O(n^{2.807})$ and $O(n^{2.3727})$ respectively.

The latest bootstrapping implementation is reported by Halevia et al. [76] which takes 6 minutes. Ducase et al. [77] improve the bootstrapping procedure to less than a second. Their improvement comes in two folds. First, they introduce a new homomorphically compute the NAND of two LWE ciphertexts. This introduces a much lower noise level than previous techniques. Second, a ring variant of the bootstrapping is utilized which reduces the asymptotic computation time of lattice cryptography from quadratic to quasi-linear. This scheme is further improved by Chillotti et al. [78] which brings the bootstrapping to only 0.1 second. It also reduces the bootstrapping key size from 1 GB to only 24 MB with the same security level.

All the past presented schemes can only perform computations over the integer. In [79], the authors introduce the first leveled homomorphic encryption scheme that allows operations over real numbers. The security assumption of CKKS scheme is based on RLWE. CKKS scheme was improved in [80] by introducing the full residue number system (RNS) variants. In [81], the authors lift the CKKS leveled scheme to FHE scheme by offering a bootstrapping function.

2.1.5 Oblivious Transfer

Aside from the works just listed, oblivious transmission is used by numerous additional secure protocols. Protocols based on GMW [82, 83] and TinyOT [47, 48, 84] use OT extensions for efficient SMC on binary circuits, and fast garbled circuit protocols utilize OT extensions in the input stage of the protocol [85]. Pinkas et al., [86,87] employ

OT extensions to provide an efficient and scalable protocol for the stated application of private set intersection. Ishai et al. [88] offer another technique for establishing malicious security based on OT. However, they only provide asymptotic complexity metrics. Furthermore, their protocol's building components, such as codes and fast Fourier transform, imply a more costly calculation than MASCOT [53], where the computation consists mostly of a few field operations. Baum et al. [89] explained enhancements to the sacrifice phase and the zero-knowledge proofs employed in SPDZ with somewhat homomorphic encryption. Their sacrifice method necessitates the generation of triples that constitute codewords. Their zero-knowledge proofs outperform Damgård et al. [11] by roughly a factor of two. In a recent development, Boyle et al. [90] propose silent OT which allows creating a large number of random, or correlated, OTs, with very little interaction. This technique offers great advantages for methods that uses edabit [28]. With a moderate increase in computing [91], the communication cost utilizing silent OT may be reduced up to 100x less than OT extension based on previous approaches [92].

2.1.6 Mixed Circuit

The above-described protocols necessitate computations on a specific mathematical structure that requires to fix the computation domain. For example, in an arithmetic circuit that favors integer calculations like addition and multiplication, the domain may carry computation modulo a prime or power of two. Another example is the modulo two calculation in binary circuits, preferred for non-linear functions like comparisons. Many applications have both linear and non-linear capabilities. Deep learning convolution layers, for example, are made up of dot products followed by a non-linear activation function. Therefore, new work of research has emerged.

ABY framework [93] (Arithmetic-Boolean-Yao) handles the semi-honest model in a two-party setting. Since then, other works have advanced the setting to differ-

ent parties and various adversary models. For example, in [94, 95] the authors offer three parties in the honest majority setting, and in [96] the authors present a dishonest majority setting with malicious security. Other literature focuses on generating compilers that can decide which part of a protocol carries in arithmetic or binary circuits [97–99].

Rotaru et al. introduce daBits technique [96] (doubly-authenticated bits) which generates random secret bits in both binary and arithmetic domains. These bits are used to convert between binary/arithmetic domains in the SMC protocols. The daBits supports any corruption setting; however, it has been utilized in the SPDZ protocol [10] under dishonest majority settings with malicious security. More efficient methods for generating daBits present in [100–102].

In late development, the function secret sharing technique is utilized for binary and arithmetic conversions and other operations such as comparison [103, 104]. Although, it relies on a trusted setup or an expensive offline phase that has not been practical for malicious adversaries. This method improves the online phase with only one round. On the other hand, edaBits provides a technique to enhance the conversion between arithmetic and binary data types in SMC [28]. The edaBits are shared integers in the arithmetic domain with a shared bit decomposition in the binary domain. When compared to daBits, edaBits can be produced more efficiently. It requires a cut-and-choose approach as well as the use of binary circuits' inherent tamper-resilient characteristics. The edaBits work well with dishonest majority protocols like SPDZ. It may, however, be applied to any corruption setup.

2.2 Preliminaries and Definitions

This section provides the commonly used notations and security definitions. It also presents the background on the secret sharing schemes used to construct the proposed

protocols. In addition to that, it introduces construction of the BFV scheme which is used to carry the secure multiplication in the proposed compiler in chapter 4.

2.2.1 Conventions and Notations

The following notations are commonly used in the literature and they are adopted for the rest of the thesis:

- $P_1, \ldots, P_n$: n parties or servers who collaboratively and securely perform the required computations.
- $\mathbb{Z}_p$: a prime domain $\{0, \ldots, p-1\}$ where p is a prime and $|p| = \ell$ represents the number of bits requires to represent p.
- $[x]$: a value x is secretly shared among the n parties. The shares are drawn from $\mathbb{Z}_p$. The domain of x is bounded by p.
- $[x]^{P_j}$ (or $[x]_p^{P_j}$): the secret share of x belongs to party P_j. Thus, $[x]$ is a set of shares denoted by $[x]^{P_1}, \ldots, [x]^{P_n}$.
- r, $r_0, \ldots, r_{\ell-1}$: r generally represents a random value in $\mathbb{Z}_p$, and $r_0, \ldots, r_{\ell-1} \in \{0, 1\}$ represent the individual bits of r where r_0 and $r_{\ell-1}$ are the least and most significant bits of r respectively.
- $s^1, \ldots, s^n$: are random values generated from $\mathbb{Z}_p$. In our proposed protocols, s^j is generated by party P_j.

The superscript/subscript in $[x]_p^{P_j}$ may be dropped for succinctness if it is clear from the context. For example, the expression below represents local computations performed by P_j based on its own shares.

- P_j: $[x]^{P_j} \leftarrow \sum_{j=1}^{n} [x^j]^{P_j}$

It produces P_j's secret share of x by summing each secret share of $[x^j]^{P_j}$. To simplify the notations, we often adopt the following expression instead:

- P_j: $[x] \leftarrow \sum_{j=1}^{n} [x^j]$

Moreover, the term "secret share" (or "secretly shared") is interchangeable with "share" (or "shared").

2.2.2 Secret Sharing and its Functionalities

This work requires that any secret sharing scheme to be used have the ability to perform the following operations, and the ones required communications among the parties are denoted as ideal functionalities with symbol $\mathcal{F}$. As stated previously, we assume that the adversary $\mathcal{A}$ is computationally bounded and control at most $n-1$ parties who remain the same throughout the protocol execution. The adversary does not learn any information about the private input of an honest party. Thus, we only need to guarantee privacy. Since we do not need to guarantee the computation correctness, the implementations of these functionalities are highly efficient. Note that detecting malicious behaviors is achieved through our proposed compiler instead of at the sub-protocol level which leads to a very efficient SC implementation against the malicious majority case.

- $\mathcal{F}_{\text{share}}(x)$: given a particular value $x \in \mathbb{Z}_p$, a dealer can generate shares $[x]^{P_1}, \ldots, [x]^{P_n} \in \mathbb{Z}_p$ of x. Each party P_j has share $[x]^{P_j}$. This must be done in a way that they can be uniquely recombined in a method applicable to the scheme to reconstruct the original value.

- $\mathcal{F}_{\text{open}}([x])$: all n shares $[x]^{P_j}$ are needed to reconstruct the original value x.

- $\mathcal{F}_{\text{mult}}([x], [y])$: given two secretly shared values x and y, it returns secret shares of xy. Specifically, the functionality returns $[xy]^{P_j}$ to party P_j.

- $\mathcal{F}_{\text{mult}_2}(\langle P_i, \alpha\rangle, \langle P_j, \beta\rangle)$: a two-party functionality that allows P_i with private input α and P_j with private input β to derive $[\alpha\beta]^{P_i}$ and $[\alpha\beta]^{P_j}$.

- Local operations: These operations does not require communication as they can be carries locally.

  - Addition with a public constant: given shares of $[x]$, and a public constant c, execute the necessary operations to calculate $[c + x]$.

  - Addition: given two shared values of $[x]$ and $[y]$, calculate the shares of the sum of the original values $[x + y]$.

  - Multiplication by a public constant: given shares of $[x]$, and a public constant c, execute the necessary operations to calculate $[cx]$.

2.3 Secret Sharing Schemes

This section will go deeper into secret sharing methods by first providing an outline of secret sharing. Then, three secret sharing schemes will be studied in further depth.

Modern cryptography is heavily predicated on the assumption that $p \neq np$. For instance, factoring integers and calculating discrete logarithms are often regarded as hard tasks on classical computers. They have been the basis for several proposed cryptosystems includes Rivest, Shamir, and Adleman's widely used RSA public key cryptosystem [105]. Another example, the Paillier cryptosystem [106] in which the security is based on the Decisional Composite Residuosity Assumption (DCRA), which has been shown to be as difficult as factoring $n = pq$, given that p and q are two large prime integers. Peter Shor illustrates a technique that can solve these problems in polynomial time using a quantum computer with a small probability of error [107]. Secret sharing systems, on the other hand, remain unaffected since their security is based on a simple mathematical observation rather than a hard problem. As we will

show, it is impossible to reconstruct a secret without a sufficiently enough number of shares.

A secret sharing scheme can be defined as two main functions as follows:

1. The first function can be implemented by the data owner (also known as a dealer) that takes some input value s (known as a secret) and break it into a set of n values (known as shares) $\{s_1, s_2, \cdots, s_n\}$, where n is the number of parties. The dealer then distributes one share to each participant. It is worth noting that these shares are random values.

2. The other function is the inverse function of the previous function. It takes a subset of shares $\{s_1, \cdots, s_t\}$ and reconstructs the original value s, where t is the threshold which represents the minimum number of shares required to rebuild a secret value.

The dealer represents the data owner, and sometimes a player may act as a dealer specifically when generating a uniformly random value with other parties. The main goal of a secret sharing scheme is to make it hard to reopen a secret value by an authorized user. Throughout this thesis, we use the words players, parties, and participants interchangeably to refer to a group of servers.

Many secret sharing have been suggested; for example, additive secret sharing, Shamir secret sharing scheme [108], replicated secret sharing, hierarchical secret sharing schemes [109], linear integer secret sharing scheme (LISS) [110], and DNF-based secret sharing. This thesis utilizes Shamir secret sharing (SSS) and replicated secret sharing (RSS) scheme for the honest majority models. On the other hand, we use additive secret sharing for the dishonest majority model.

In this work the participants are divided into two groups: A qualified group which is a set of player that can reconstruct a secret from a given set of shares. The other group is a forbidden group which is a set of players that cannot reconstruct a secret

given a set of shares. The group of all the qualified subsets is known as the access structure of the scheme. These schemes also known as t out of n secret sharing schemes (t, n). Where n represents the number of players and t refers to the scheme threshold which simply means that the qualified players requires at least t shares to reconstruct a given secret.

2.3.1 Additive Secret Sharing Scheme (ASS)

The additive secret sharing scheme is a straightforward scheme. The secret values are shared as the sum of the shares, thus it requires only addition modulo p to reconstruct a secret. It is worth noting that the modulo operation is in the core of the scheme to guarantee a uniformly random values. It is obvious that additive secret sharing necessitates the use of all shares in order to reveal the secret; hence, it is also known as the full threshold scheme.

Sharing

A dealer can deal a secret value s by selecting $n-1$ random values which represents the shares of the first $n-1$ participants. The last shared value $[s]^{P_n}$ is then obtained by summing the previous $n-1$ random values and then subtract the summed value for the secret s. The dealer then can send each share to a designated party. This can be formulated as in equation 2.1, where $[s]^{p_i}$ are random values in the defined field.

$$[s]^{p_n} = s - \sum_{i=1}^{n-1} [s]^{p_i} \mod p \tag{2.1}$$

Reconstructing

Under the additive secret sharing scheme, all the shares are needed to open up a secret value. Therefore, all the parties require to send their shares to an authorized

user which then can sum up the shares and carry modulo p as shown in equation 2.2.

$$s = \sum_{i=1}^{n} [s]^{p_i} \mod p \tag{2.2}$$

Security

The additive secret sharing scheme offers a perfect security. The security of the scheme follows the fact the every party gets a random value. Indeed, this offers a form of one-time pad. The scheme promises a high security in the sense that all the shares are required to reconstruct a secret. Thus, the scheme fits the semi-honest model requirements where the participants follow the protocol steps. However, the scheme has a disadvantage. For example, parties may occasionally face hardware/software errors and be out of the service. In some cases this could happen as a form of denial of service (DoS) attack. In this case, under the additive secret sharing scheme, the rest of the parties cannot progress further and the whole protocol maybe terminated. In other words, the additive secret sharing scheme has no fault tolerance mechanism due to the fact that it requires all the shares at the end of the protocol to get the output reconstructed correctly.

2.3.2 Shamir Secret Sharing Scheme (SSS)

Shamir secret sharing proposed in 1979 [108]. The scheme still widely used due to its useful properties. The dealer can generate a uniformly random polynomial of $t-1$ degree such that the secret value is kept with the rest of the coefficients. Then each point on that polynomial represents a share of the secret. The coefficients, secret, and shares all live in the same finite prime field $\mathbb{F}_P$ of a sufficient prime p. Modulo operation is carried throughout all the scheme operations similar to the additive secrete sharing scheme. The scheme allows to select a polynomial degree

that is independent from the number of participant parties. Thus, the scheme forms a natural threshold sharing scheme where the number of shared that are required to reconstruct a secret has to be at least t. Reconstructing a secret can be done easily using different methods for example Lagrange polynomial interpolation and Van der Monde matrix.

Sharing

A secret value s is shared under Shamir secret sharing scheme via generating a polynomial of degree $t-1$ as in equation 2.3.

$$f(x) = s + \alpha_1 x + \alpha_2 x^2 + \cdots + \alpha_{t-2} x^{t-2} + \alpha_{t-1} x^{t-1} \tag{2.3}$$

The dealer evaluates the polynomial for different points based on the participants identifier $id_j \in \{1, \cdots, n\}$ such that each party p_j associated with a unique identifier id_j. Thus, the share of each party is computed as:

$$[s]^{p_j} = f(id_j) \tag{2.4}$$

Recombining

Reconstructing a secret requires that at least t shares to be sent to an authorized user. The authorized user can interpolate the polynomial which can be efficiently computed using different methods such as Van der Monde matrix or Lagrange interpolation. Here we present Lagrange interpolation, since the focus is on the y intercept then:

$$f(x) = \sum_{i=1}^{n} y_i \prod_{j=1; j \neq i}^{n} \frac{x - x_j}{x_i - x_j} \tag{2.5}$$

The above equation can be simplified further since the case of interest to open the secret is $f(0)$.

$$s = f(0) = \sum_{i=1}^{n} y_i \prod_{j=1; j \neq i}^{n} \frac{-x_j}{x_i - x_j} \tag{2.6}$$

Security

Recall that to reopen a secret, the scheme requires t shares. Thus it is impossible to reconstruct the polynomial given any $t-1$ shares. The security of Shamir secret sharing scheme is based on the fact that a function of degree $t-1$ over $\mathbb{Z}_p$ cannot be reconstructed with less than t points on the function. The security is information-theoretic as all secret values s are equally likely given only $t-1$ points on the function.

Multiplication under SSS

Unfortunately, Share multiplication cannot be computed using the linear property of the transformation since multiplying two polynomials of the same degree yields a polynomial with a degree double that of the original polynomials. This means that a t-out-of-n threshold must be use such that $n \geq 2t + 1$ [111]. The parties compute their local share multiplication, which corresponds to the polynomial that stores the secret product. However, the number of shares necessary to rebuild this polynomial is usually greater than n. As a result, it is not easy to rebuild the product secret. Instead, parties reshare the secret held in the product polynomial under the same original polynomial degree. This is accomplished by performing a degree reduction step and then resharing the local share result. Thus, each party produces a random polynomial with the same source degree as the dealer and follows the same steps when dealing a secret. Before exchanging the last shares, each party constructs a vector of Lagrange coefficients or the first row of an appropriate Van der Monde matrix. Then, each party multiplies the value associated with their index by the shares of

their product. Finally, each participant adds up the shares to get the shares for the required degree of the product. Protocol 1 shows the full steps of Shamir secret sharing multiplication.

Protocol 1 Mult($[x][y]$) $\rightarrow [c] = [x \cdot y]$

Require: p is prime of ℓ bits, β^j generated using equation 2.6, where $1 \leq j \leq n$

1: Each party P_j:

(a) computes $[u]^j = [x]^j[y]^j$

(b) deals shares of $[u]^j$ to $[u']^{j1}, \cdots, [u']^{jn}$

(c) computes $[w]^{j1} = [u']^{j1}\beta^j, \cdots, [w]^{jn} = [u']^{jn}\beta^j$

(d) sends each $[w]^{jk}$ to P_k where $j \neq k$

(e) receives $[w]^{kj}$ from other parties

(f) computes $[c]^j = [u']^{jj} + \sum_{\substack{k=1 \\ j \neq k}}^{n} [w]^{kj}$

The protocol is secure under the assumption that all parties follow the steps of the protocol faithfully. For detailed explanation regarding security and correctness we refer the reader to [112]. A full example is present in the appendix B.2 which shows a step by step how to multiply two secretly shared values using Shamir secret sharing scheme.

2.3.3 Replicated Secret Sharing scheme (RSS)

Replicated secret sharing also known as replicated additive secret sharing scheme allows lifting the restriction on the threshold of the additive secret sharing scheme [113]. Such that it supports a threshold $(t < n - 1, n)$ rather than being restricted to only fully threshold as describes in section 2.3.1. It also promises faster multiplication by only sending a single field element per multiplication gate. In this arrangement, several parties own multiple shares of a secret; hence the name replicated. The scheme benefits a limited number of participants. However, due to the large number

of subsets required by the scheme, it does not scale effectively in the case of a large number of parties. As a result, it suffices for the semi-honest and malicious minority models, both of which require three participants. Since we are utilizing replicated secret sharing scheme for semi-honest/honest majority, we restrict the scheme for three parties following the same practice in state of the art library (MP-SPDZ) [29].

Sharing

As in the additive secret sharing scheme, a dealer can share a secret $s \in \mathbb{F}$ by selecting random elements $s_j \in \mathbb{F}$, such that $s = \sum_{j=1}^{n} s_j$ for n parties. Then the dealer distributes the shares such that each party P_j holds $n-1$ shares. In the case of three parties, for example, parties P_1, P_2, and P_3 hold (s_1, s_3), (s_2, s_1), and (s_3, s_2) consecutively.

Recombining

To reconstruct a secrete s from its shares, in the case of three parties, party P_j, P_{j-1} and P_{j+1} send s_j s_{j-1}, and s_{j+1} respectively to an authorized user. The authorized user can sum up the shares and get the secret back.

Security

The protocol's security mirrors the fact that members of every unqualified set miss exactly one additive share. On the other hand, a qualified set cannot be included in any unqualified set; Therefore, members of a qualified set jointly view all shares and can thus reconstruct s.

Multiplication under RSS

Given $[a]$ and $[b]$, two secretly shared values based on the replicated sharing scheme, our goal is to compute their product such that each party holds a share $[c]$ where $c = a \cdot b$. Since we only require this scheme under the semi-honest/honest majority settings which only needs three parties, we will restrict the protocol to this case. The steps are as follows:

1. The parties generate a random correlated value r^j, such that $\sum_{j=1}^{n} r^j = 0$. Generating this random value is a straightforward process, we refer the reader to the detailed treatment given in [114].

2. Each party P_j locally computes the pairwise multiplication of all the possible combinations and add his local share of r^j, then send the result to the next party.

3. Each party outputs the final shares as a pair of values represent the locally calculated product and the received value.

Protocol 2 Mult$([x][y]) \rightarrow [c] = [x \cdot y]$

Require: p is prime of ℓ bits, $1 \leq j \leq n$, and r^j such that $\sum_{j=1}^{n} r^j = 0$

1: P_j: computes $[u]^j = [x]^j[y]^j + [x]^j[y]^{j-1} + [x]^{j-1}[y]^j + r^j$ and sends it to P_{j+1}
2: P_j: receives $[u]^{j-1}$
3: P_j: output $[c] = ([u]^j, [u]^{j-1})$

Correctness follows the pairwise multiplication of additive values such that:

$$x \cdot y = (x1 + x2 + x3)(y1 + y2 + y3) \tag{2.7}$$

Recall that $[u]$ values hold the product of xy as in protocol 2. This gives $x \cdot y = u^1 + u^2 + u^3 = \sum_{j=1}^{3}(x^j y^j + x^j y^{j-1} + x^{j-1} y^j) + \sum_{j=1}^{3} r^j$. Given that $\sum_{j=1}^{3} r^j = 0$, hence, equation 2.7 holds.

The protocol is secure under the semi-honest adversary model. The full proof is presented in [114].

2.4 BFV Encryption Scheme

In this section, we provide a brief introduction to the BFV encryption scheme [115]. We utilize BFV in our design to compute a pairwise multiplication that takes place between two pairs of parties under the malicious majority setup. The design requires a circuit of one multiplication level. It is worth mentioning that SPDZ relies on BGV scheme which has the same performance as the BFV under one level of multiplication.

BFV relies on the polynomial ring $R = \mathbb{Z}[x]/(f(x))$. Where $f(x) \in \mathbb{Z}[x]$ is a monic irreducible polynomial of degree $d = 2^n$. Elements of the ring R is denoted in lowercase bold, $\mathbf{a} \in R$ such that $\mathbf{a} = \sum_{i=0}^{d-1} a_i \cdot x^i$. Where a_i is the coefficients of an element $\mathbf{a} \in R$.

Distributions Required by BFV

BFV requires to sample elements from different distributions as follows:

- $D_{\mathbb{Z},\sigma}$: Discrete Gaussian distribution, which assigns a probability proportional to $exp(-\pi|x|^2/\sigma^2)$ to each $x \in \mathbb{Z}$.
- χ: This distribution generates an element on R based on $D_{\mathbb{Z},\sigma}$.

One Level BFV Scheme

In our design, the plaintext space is R_p for some integer modulus which does not require to be a prime. However, to enable the batching then the plaintext modulus must be a prime number such that $p \equiv 1 \mod 2N$, where N is the polynomial modulus degree [30]. The ciphertext space is R_q, where q is the ciphertext modulus.

Let $\Delta = \lfloor q/p \rfloor$. The BFV scheme is given by three algorithms $\{KeyGen, Enc, Dec\}$. The algorithms are parametrized by a computational security parameter λ which we set at 128. The BFV algorithms are as follows:

- KeyGen(1^λ): samples $\mathbf{s} \leftarrow R_2$, set the secret key $sk = \mathbf{s}$, samples $\mathbf{a} \leftarrow R_q, \mathbf{e} \leftarrow \chi$. Sets $\mathbf{b} = [-(\mathbf{a} \cdot \mathbf{s} + \mathbf{e})]_q$, outputs $pk = (\mathbf{b}, \mathbf{a})$.
- Enc(pk, $\mathbf{m}$): to encrypt $m \in R_p$, sample $\mathbf{u} \leftarrow R_2$, $\mathbf{e}_1, \mathbf{e}_2 \leftarrow \chi$, set $\mathbf{c}_0 = [\mathbf{b} \cdot \mathbf{u} + \mathbf{e}_1 + \Delta \cdot \mathbf{m}]_q$, $\mathbf{c}_1 = [\mathbf{a} \cdot \mathbf{u} + \mathbf{e}_2]_q$. Output $ct = (\mathbf{c}_0, \mathbf{c}_1)$.
- Dec($\mathbf{s}$, ct): output $m = [\lfloor \frac{p \cdot [\mathbf{c}_0 + \mathbf{c}_1]_q}{q} \rceil]_p$

Homomorphic Operations

The elements of a ciphertext ct can be treated as the coefficients of a polynomial $ct(x)$. Evaluating this polynomial for $\mathbf{s}$ we obtain $[ct(\mathbf{s})]_q = \Delta \cdot \mathbf{m} + \mathbf{v}$, where $\mathbf{v}$ is the noise contained in the ciphertext. Using this interpretation, then

Given two ciphertexts $ct_1 = (\mathbf{c}_{10}, \mathbf{c}_{11}), ct_2 = (\mathbf{c}_{20}, \mathbf{c}_{21})$. The homomorphic operations are as follows:

- Addition: set $\mathbf{c}_0 = [\mathbf{c}_{10} + \mathbf{c}_{20}]_q$, $\mathbf{c}_1 = [\mathbf{c}_{12} + \mathbf{c}_{21}]_q$. We define $Add(ct_1, ct_2) = (\mathbf{c}_0, \mathbf{c}_1)$.
- Multiplication: Set $\mathbf{c}_0 = [\lfloor \frac{p \cdot (\mathbf{c}_{10} \cdot \mathbf{c}_{20})}{q} \rceil]_q$,

  $\mathbf{c}_1 = [\lfloor \frac{p \cdot (\mathbf{c}_{10} \cdot \mathbf{c}_{21} + \mathbf{c}_{11} \cdot \mathbf{c}_{20})}{q} \rceil]_q$, $\mathbf{c}_2 = [\lfloor \frac{p \cdot (\mathbf{c}_{11} \cdot \mathbf{c}_{21})}{q} \rceil]_q$. We define $Mult(ct_1, ct_2) = (\mathbf{c}_0, \mathbf{c}_1, \mathbf{c}_2)$.
- Relinearize: it reduces the three components ciphertext into two components ciphertext. The idea is to slice $\mathbf{c}_2$ into parts of small norm based on W such that $\mathbf{c}_2 = \sum_{i=0}^{\ell} W^i \cdot \mathbf{c}_2^{(i)} \mod q$. For $i \in \{0, \cdots, \ell\}$, and $\ell = \lfloor \log_w q \rfloor$. The
  - EvaKeyGen($\mathbf{s}$, w): it generates a relinearization key by sampling $\mathbf{a}_i \leftarrow R_q$, $\mathbf{e}_i \leftarrow \chi$.

Output $rlk = ([-(\mathbf{a}_i \cdot \mathbf{s} + \mathbf{e}_i) + W^i \cdot \mathbf{s}^2]_q, \mathbf{a}_i)$

- Relinearize(ct, rlk): write $\mathbf{c}_2 = \sum_{i=0}^{\ell} \mathbf{c}_2^{(i)} \cdot W^i$, where the coefficients of $\mathbf{c}_2 \in R_W$, set $\mathbf{c}'_0 = [\mathbf{c}_0 + \sum_{i=0}^{\ell} rlk[i][0] \cdot \mathbf{c}_2^{(i)}]_q$ and $\mathbf{c}'_1 = [\mathbf{c}_1 + \sum_{i=0}^{\ell} rlk[i][1] \cdot \mathbf{c}_2^{(i)}]_q$. Return $(\mathbf{c}'_0, \mathbf{c}'_1)$

Ciphertext Noise

Given that the distribution is bounded by B such that $||\chi|| < B$. The noise associated with L levels of multiplication is roughly of size $2 \cdot B \cdot \delta_R^{2L+1} \cdot p^L$, where δ_R is the expansion factor of R which is defined as $\delta_R = max\{||\mathbf{a} \cdot \mathbf{b}||/(||\mathbf{a}|| \cdot ||\mathbf{b}||) : \mathbf{a}, \mathbf{b} \in R\}$. Since we only require one level of multiplication this leads to noise around $2 \cdot B \cdot \delta_R^3 \cdot p$. The decryption works correctly as long as the ciphertext noise is smaller than $\Delta/2$.

Chapter 3

Secure Comparison Protocols

3.1 Introduction

Secure multiparty computing (SMC) is a sub field of cryptography that encompasses many techniques. Generally, it enables a group of distrustful parties to compute a function $f(x)$ without exposing the input x of other parties. SMC developed as a theoretical paradigm in the early 1980s. Some researches focus on customizing SMC solutions to a specific situation in order to compensate for the efficiency disadvantage. Other efforts aim to improve the efficiency of core SMC building blocks, which may be used in a wide range of applications. Thus, research has progressed from theoretical to practical paradigms [29, 116]. Recent studies demonstrate that SMC can solve real-world issues. Several organizations now provide SMC solutions [21, 22].

Secure comparison (SC) is one of the core SMC building blocks. It is essential in numerous applications, including online auctions, large data analytics, machine learning, and query processing. Andrew Yao initially presented the secure comparison problem, often known as the millionaires' problem, in 1982 [1]. Since then, many research efforts have been directed to increase SC's efficiency in different circuit do-

mains such as arithmetic and Boolean circuits. In a recent development, research shifted to a mixed-mode where nonlinear operations (comparisons) can be carried out efficiently using Boolean circuits. The arithmetic circuit, on the other hand, can compute linear operations (addition and multiplication). Nonetheless, despite advancements in SC protocols, it remains a bottleneck for privacy-preserving computation. As a result, every advancement in this area has a considerable impact on the total performance of privacy-preserving applications.

In this chapter, we propose an efficient, secure comparison protocol which offer information-theoretic security. The protocol named SC_{3P} is designed in the case of no collusion between the parties. As a result it works for semi-honest adversaries and may fit the honest majority setting. The protocol works for 3 parties and requires no secure multiplication which represents the expensive part of SMC operations.

In the case of the dishonest majority setting, constructing an efficient secure comparison protocol is a cumbersome problem. Therefore, we designed a protocol utilizing the best practice in the literature to gain the best efficiency. The protocol works with collusion up to $n-1$ parties and works for any number of parties. The protocol improves upon the state-of-the-art secure comparison protocol Rabbit. It requires to invoke only one random value r associated with its random bits r_i. While Rabbit requires two random values. Moreover, the protocol only invokes the secure sub protocol (bit less than) once, whereas Rabbit needs three invocations of the same sub protocol. Finally, it is worth noting that these protocols may produce random values during a preprocessing step, making them appealing for real-world applications, as these are critical factors in developing practical secure systems.

3.2 Comparison Reduction

Directly comparing two secretly shared values is possible [7], but it is computationally expensive. Instead, we transform comparison between two secretly shared values into comparison between a secretly shared and a publicly known value. Our implementation of a semi-honest secure comparison protocol is inspired by the ideas proposed in [8, 9].

Suppose a and b are non-negative integers share secretly shared among n parties: $P_1, \ldots, P_n$. Let $[a]$ and $[b]$ be the secret shares of a and b. Both values are bounded by $\lfloor \frac{p}{2} \rfloor$, p is a prime and $\ell = \lceil \log_2 p \rceil$. Let $c = 2a - 2b \bmod p$ and c_0 and c_ℓ denote the least and most significant bits of c respectively. The following observation holds:

$$c_0 = \begin{cases} 0 & \text{if } a \geq b \\ 1 & \text{if } a < b \end{cases} \tag{3.1}$$

When $a \geq b$, $c = 2a - 2b$ must be an even number. Thus, $c_0 = 0$. On the other hand, when $a < b$, $c = p - (2b - 2a)$. Since p is an odd number and $2b - 2a$ is even, c must be odd. That is, $c_0 = 1$. It is clear that c_0 holds the comparison result between a and b. Next we show how c_0 can be derived. Let $\eta = c + r \mod p$ where r is randomly selected from $\mathbb{Z}_p$. Then the comparison result depends on the following:

$$c_0 = \begin{cases} \eta_0 \oplus r_0 & \text{if } \eta \geq r \\ 1 - \eta_0 \oplus r_0 & \text{if } \eta < r \end{cases} \tag{3.2}$$

Instead of comparing two secretly shared values a and b, now the comparison can be performed between a public value η and a secretly shared value r. If r is randomly chosen and remains secret, disclosing η does not reveal any information regarding a and b. In addition, because the parties obtain the secret shares of the individual bits of r while generating r, the comparison is simplified. However, it is not straightforward

to efficiently generate a secretly shared r and its bits within a domain defined by a prime number. Next we present how to produce such r efficiently in practice.

3.3 Building Blocks

In this section, we introduces a number of primitives required below. Some of these sub-protocols are given in [55], however, are introduced here to provide a detailed analysis as well as for completeness. Most of these protocols are related to the generation of random values unknown to all parties. It is worth mentioning that some of the protocol may fail. However, this does not compromise the privacy of the inputs. Failure here simply means to the inability to generate a proper random value, however, it is detected. In general, the probability of failure will be of the order $1/p$ where p is a prime number.

3.3.1 Generating Secretly Shared Random Values

The proposed protocols require the parties to secretly share a random bit or a random value from $\mathbb{Z}_p$. We define these functionalities below and their implementations.

- $\mathcal{F}_{\text{rand}}(p)$: generating a random value r in $\mathbb{Z}_p$ and secretly sharing it among the n parties. At the end, P_j holds share $[r]^{P_j}$, and no parties know r.

- $\mathcal{F}_{\text{rand}_b}(p)$: generating a random bit $\tau \in \{0, 1\}$ and secretly sharing it among the n parties. At the end, P_j holds share $[\tau]^{P_j}$, and no parties know τ.

In both functionalities, p also defines the domain for the random shares. Protocol 3 implements $\mathcal{F}_{\text{rand}}$, and Protocol 4 implements $\mathcal{F}_{\text{rand}_b}$. At step 3 of Rand_b, $\mathcal{F}_{\oplus}$ represents a functionality that produces shares of $[w^1 \oplus \cdots \oplus w^n]$ which results shares of $[w]$. To implement $\mathcal{F}_{\oplus}$, a square-root based method is proposed in [7] which is efficient for certain prime modulus such as Mersenne prime. However, in our implementation,

the modulo provided in the SEAL library [30] are not efficient for perform square-root operations. Thus, we use the $\mathcal{F}_{\text{mult}}$ functionality to implement $\mathcal{F}_{\oplus}$. Since $\alpha \oplus \beta \equiv \alpha + \beta - 2\alpha\beta$, a straightforward way for achieving $\mathcal{F}_{\oplus}$ is to apply $\mathcal{F}_{\text{mult}}$ pairwise. According to our empirical results, this approach actually provides better efficiency in practice.

Protocol 3 $\text{Rand}(p) \rightarrow [s]$

Require: p is prime of ℓ bits, and $1 \leq j \leq n$

1: P_j: selecting a random value $s^j \in \mathbb{Z}_p$
2: P_j: $\mathcal{F}_{\text{share}}(s^j)$
3: P_j: $[s]^{P_j} \leftarrow \sum_{i=1}^{n} [s^i]^{P_j}$

Protocol 4 $\text{Rand}_b(p) \rightarrow [w]$

Require: p is prime of ℓ bits, and $1 \leq j \leq n$

1: P_j: selecting a random bit $w^j \in \{0, 1\}$
2: P_j: $\mathcal{F}_{\text{share}}(w^j)$
3: P_j: $[w]^{P_j} \leftarrow \mathcal{F}_{\oplus}([w^1], \ldots, [w^n])$

It is worth noting that the complexity of protocol 3 is considered as one multiplication invocation which is done in one round. However, when considering dishonest majority setting the protocol cost becomes negligible since there is no multiplication. On the other hand, the complexity of protocol 4 is of n multiplication invocations which can be done in $\log n$ rounds.

3.3.2 Generating Random Bitwise Shared Value

To generate a uniformly random bitwise shared value $[r]$, and its bit-wise shares $[r_0], \ldots, [r_{\ell-1}]$ such that $r \in \{0, \ldots, p-1\}$ and $r_i \in \{0, 1\}$ for $0 \leq i \leq \ell - 1$, we follow the ideas presented in [117]. The parties first generate ℓ random shared bits $[r_i]$, and then verify if $r = \sum_{i=0}^{\ell-1} 2^i r_i < p$ by computing a vector of elements$[e_i]$ defined by Equation 3.3:

$$[\hat{e}_i] = [s_i] \left(1 + \hat{p}_i - [r_i] + \sum_{j=i+1}^{\ell-1} \hat{p}_j \oplus [r_j] \right) \tag{3.3}$$

where $[s_i] \in \mathbb{F}$ is a random shared value, $\hat{p} = p - 1$, and $\hat{p}_i$ represents the i^{th} bit of $\hat{p}$. The parties then reveal the shares of $[\hat{e}_i]$ and determine $\hat{p} < [r]$ if $\exists \hat{e}_i = 0$. If none of the $\hat{e}_i$ values are 0, the parties derive $[r] \leftarrow \sum_{i=0}^{\ell-1} 2^i [r_i]$. Protocol 5 shows the full steps of generating a uniformly random bit-wise shared value in $\mathbb{Z}_p$:

Protocol 5 Gen_Bitwise_Shares$(p) \rightarrow [r], [r_0], \ldots, [r_{\ell-1}]$

Require: p is prime of ℓ bits and $\hat{p} = p - 1$.

1: The parties collaboratively generate ℓ secretly shared random values $[s_0], \ldots, [s_{\ell-1}]$ where s_i in $\mathbb{Z}_p$:
   - $[s_i] \leftarrow \text{Rand}(p)$, for $0 \leq 1 \leq \ell - 1$

2: The parties collaboratively generate ℓ secretly shared random bits $[r_0], \ldots, [r_{\ell-1}]$, where $r_i \in \{0, 1\}$:
   - $[r_i] \leftarrow \text{Rand}_2(p)$, for $0 \leq 1 \leq \ell - 1$

3: Based on Equation 3.3 and the shares, the parties securely verify if r is in $\{0, \ldots, p-1\}$:
   (a) $[\hat{e}_i] \leftarrow [s_i] \left(1 + \hat{p}_i - [r_i] + \sum_{j=i+1}^{\ell-1} \hat{p}_j \oplus [r_j]\right)$, for $0 \leq i \leq \ell - 1$.
   (b) $\hat{e}_i \leftarrow \mathcal{F}_{\text{open}}([e_i])$, for $0 \leq i \leq \ell - 1$.
   (c) If $\exists \hat{e}_i = 0$, go back to Step 1 and repeat the protocol.

4: Each party locally derives $[r] \leftarrow \sum_{i=0}^{\ell-1} 2^i [r_i]$, and having $[r], [r_0], \ldots, [r_{\ell-1}]$ as the party's output.

- Step 1: The purpose of this step is to generate the ℓ randomly shared values $[s_0], \ldots, [s_{\ell-1}]$ in $\mathbb{Z}_p$, used in Equation 3.3 to prevent leaking information regarding r.

- Step 2: Each party randomly selects ℓ bits and secretly shares these bits with the other parties. From these shares, the party produces shares of $[r_0], \ldots, [r_{\ell-1}]$.

- Step 3: This step verifies if the r value whose ℓ bits are represented by $r_0, \ldots, r_\ell$ are less than p, where r_0 and r_ℓ represent the least and most significant bits respectively. If this verification fails (one or more $\hat{e}_i$ values are zero), the protocol restarts from Step 1. At step (b), $\mathcal{F}_{\text{open}}$ is a functionality that reconstructs the actual value from its secret shares.

- Step 4: After the previous verification is passed, the protocol returns the shares of $[r]$ along with the shares of its individual bits: $[r_0], \ldots, [r_{\ell-1}]$.

It is worth mentioning that Step 3 of Protocol 5 seems to leak information about the random bit $[r_i]$ when $\hat{e}_i = 0$ since $\hat{p}$ is known to each party. However, when this happens, the protocol restarts. As a result, the protocol does not leak any information regarding r and its individual bits.

The complexity of generating random bitwise shared values consists of generating the ℓ bits which requires $\ell(n-1)$ multiplication invocations in $\log n$ rounds. The protocol also makes use of ℓ masks. Moreover, it requires ℓ multiplications to perform the masking. Overall this consists ℓn multiplications in $2 + \log n$ rounds considering $[r_i]$ and $[s_i]$ to be generated in parallel.

3.3.3 The Not_Zero Protocol

Given a secretly shared value $[v]$ where $v \in \{0, \ldots, \ell\}$, the Not_Zero protocol returns $[1]$ if $v \neq 0$; otherwise, it returns $[0]$. To implement that protocol, we adopt the symmetric function idea presented in [7]. Let define an ℓ-degree polynomial $\phi(x)$ over $\mathbb{Z}_p$, such that

$$\phi(x) = \begin{cases} 0 & \text{if } x = 1 \\ 1 & \text{if } x \in \{2, \ldots, \ell + 1\} \end{cases} \tag{3.4}$$

To securely evaluate $\phi(x)$ on $v + 1$ gives us the desired result. The detail steps are provided in Protocol 6:

- Step 1: One designed party can produce $\phi(x)$ by using Lagrange interpolation in $\mathbb{Z}_p$, and share $\phi(x)$ with the other parties.

- Step 2: At step (a), the parties securely perform i^{th} exponentiation of $v+1$ using exponentiation by squaring method whenever possible. For example, the parties

perform secure multiplication functionality $\mathcal{F}_{\text{mult}}([v+1],[v+1])$ to produce $[(v+1)^2]$, $\mathcal{F}_{\text{mult}}([v+1],[(v+1)^2])$ to produce $[(v+1)^3]$, $\mathcal{F}_{\text{mult}}((v+1)^2],[(v+1)^2])$ to produce $[(v+1)^4]$, etc. Then at step (b), these shares are used to construct shares of $[\phi(v+1)]$.

Protocol 6 Not_Zero$([v]) \rightarrow [\phi(v+1)]_p$

Require: p is a prime and $[v]$, where $v \in \{0,\ldots,\ell\}$

1: The parties agree on an ℓ-degree polynomial defined in Equation 3.4: $\phi(x) \leftarrow \left(\sum_{i=0}^{\ell} a_i x^i\right) \bmod p$

2: Securely compute $[\phi(v+1)]$:

(a) $[z_i] \leftarrow [(v+1)^i]$ for $0 \le i \le \ell$

(b) $[\phi(v+1)] \leftarrow \sum_{i=0}^{\ell} a_i[z_i]$

The complexity of protocol 6 can be carried in 3 rounds and 5ℓ multiplication invocations. We refer the reader to [9] for detailed analysis.

3.4 Secure Comparison Assuming no Collusion

In this section, we present an efficient secure comparison protocol SC_{3P} which is tailored for semi-honest settings. The protocol requires at least three parties and is not limited to this case. However, it performs efficiently with small number of participants. Recall the comparison reduction in section 3.2, where the comparison of two secretly shared values $[a],[b]$ is reduced to compare a publicly known value η and a secretly shared random value r.

An alternative approach to compare η and r is the arithmetic comparison circuit $\mathbb{C}$, first suggested in [55]. Here we presents the steps of the protocol for analysis and completeness.

It is easy to verify h retains the most significant bit j of η where η and r are different; that is, $\eta_j \neq r_j$ and $\eta_i = ri$, for $j < i \le \ell - 1$. The existing research has

Protocol 7 $\mathbb{C}(\eta, r) \rightarrow h \in \{0, 1\}$

Require: $p = 2^{\ell} - 1$ and $0 \leq i \leq \ell - 1$

1: $e_i \leftarrow \eta_i \oplus r_i$
2: $f_i \leftarrow \vee_{j=i}^{\ell-1} e_j$
3: $g_{\ell-1} \leftarrow f_{\ell-1}$
4: $g_i \leftarrow f_i - f_{i+1}$, for $i = \ell - 2$ to 0
5: $h = \sum_{i=0}^{\ell-1} g_i \eta_i$

been focusing on developing efficient secure methods to compute f_i. The approaches given in [55] use a symmetric function that maps input from $1, ..., \ell+1$ to $\{0, 1\}$. The function can be represented as a polynomial which can be evaluated securely.

In this section, we present a new and more efficient way to compute f without using the symmetric function. Also, as, the group delimiter p in our proposed protocol is an odd positive integer and not necessarily a prime. Therefore, secure polynomial evaluation is not applicable in our design.

To securely compare r and η, let $e = r \oplus \eta$. We calculate a special vector as in equation 3.5.

$$\gamma_i = \tau_i \left(1 + \eta_i - r_i + \sum_{k=i+1}^{\ell-1} e_k \right) \mod N_1 \tag{3.5}$$

We depend on the following observation:

- If $\forall i, \gamma_i \neq 0$, then $\eta \geq r$.
- If $\exists i, \gamma_i = 0$, then $\eta < r$.

Note that γ_i is either 0 or uniformly distributed in $Z_{N_2}^*$ where $N_2 = \ell + 1$. In the second condition, the index i is also unique.

To achieve the best efficiency, our comparison protocol is asymmetric and work by designating a party to facilitate secret sharing based secure computation. Therefore, the protocol is secure in the semi-honest model and may work in the honest majority setting as well. Since one of the parties is appointed; thus, the other two parities need to protect their input before sending any value to that party. Therefore, One of the two parities generates the following random values and share it with the second

party.

- $\tau_i \in \mathbb{Z}^*_{N_2}$: a random vector of ℓ entries to randomize the intermediate result.
- $\delta \in \{0, \cdots, \ell - 1\}$: a random value to shift the index of the intermediate vector result.
- $\zeta \in \{0, 1\}$: a random bit which is used to flip the comparison such that if $\zeta = 0$ then the comparison stays the same $\langle \eta, [r] \rangle$, otherwise, the comparison flips to $\langle [r], \eta \rangle$.

The protocol is defined as follows: $\text{SC}_{3P}(\langle P_1, [a]^{P_1}_{N_1}, [b]^{P_1}_{N_1} \rangle, \langle P_2, [a]^{P_2}_{N_1}, [b]^{P_2}_{N_1} \rangle, \langle P_3, \perp \rangle)$ $\rightarrow (\langle P_1, [c_0]^{P_1}_{N_1} \rangle, \langle P_2, [c_0]^{P_2}_{N_1} \rangle)$
Where a and b are secretly shared between P_1 and P_2, and it returns two shares $[c_0]^{P_1}_p$ and $[c_0]^{P_2}_p$ respectively to P_1 and P_2. The key steps of the protocol is given in Algorithm 8.

Through out the protocol, the index i varies from 0 to $\ell - 1$. The parties are indexed by j, the protocol works as follows:

1. P_3 generates a random number r and its shares in $\mathbb{Z}_{N_1}$, and sends the shares of r to P_1 and P_2 respectively.
2. P_1 and P_2:
   (a) Use the shares of r to disguise c and send the result to the other party.
   (b) Reveal $\eta = c + r$ and compute secret shares of $e = \eta \oplus r$.
   (c) Derive secret shares of γ_i, as defined in Equation 3.5 step (e).
   (d) Randomly shift the secret shares of γ_i and send them to P_3.
3. P_3 reconstructs $\hat{\gamma}_i$, derives the randomized comparison results f, and sends shares of f to P_1 and P_2.
4. P_1 and P_2 de-randomize f to derive shares of c_0.

Protocol 8 $\mathrm{SC}_{3P}(\langle P_1, [a]_{N_1}^{P_1}, [b]_{N_1}^{P_1}\rangle, \langle P_2, [a]_{N_1}^{P_2}, [b]_{N_1}^{P_2}\rangle \langle P_3, \perp\rangle) \rightarrow$
$(\langle P_1, [c_0]_{N_1}^{P_1}\rangle, \langle P_2, [c_0]_{N_1}^{P_2}\rangle)$

Require: $N_1 = 2^\ell - 1$ defines the domain of $[a]$ and $[b]$, $N_2 = \ell + 1$, and $0 \leq i \leq \ell - 1$

1: P_3
   (a) Generate $r \in_R \mathbb{Z}_{N_1}$, $[r]_{N_1}^{P_j}$, and $[r_i]_{N_2}^{P_j}$, for $j \in \{1, 2\}$
   (b) Send $[r]_{N_1}^{P_j}$ and $[r_i]_{N_2}^{P_j}$ to P_j

2: P_1
   (a) $[c]_{N_1}^{P_1} \leftarrow 2\left([a]_{N_1}^{P_1} - [b]_{N_1}^{P_1}\right)$
   (b) $[\eta]_{N_1}^{P_1} \leftarrow [c]_{N_1}^{P_1} + [r]_{N_1}^{P_1}$
   (c) Generate $\tau_i \in_R \mathbb{Z}_{N_2}^*$, a random shift $\delta \in \{0, \ldots, \ell - 1\}$, and a random bit ζ
   (d) Send $[\eta]_{N_1}^{P_1}$, τ_i, δ and ζ to P_2

3: P_2
   (a) $[c]_{N_1}^{P_2} \leftarrow 2\left([a]_{N_1}^{P_2} - [b]_{N_1}^{P_2}\right)$
   (b) $[\eta]_{N_1}^{P_2} \leftarrow [c]_{N_1}^{P_2} + [r]_{N_1}^{P_2}$
   (c) Send $[\eta]_{N_1}^{P_2}$ to P_1

4: P_j $(j \in \{1, 2\})$
   (a) $\eta \leftarrow [\eta]_{N_1}^{P_1} + [\eta]_{N_1}^{P_2}$
   (b) $[e_i]_{N_2}^{P_j} \leftarrow (j-1)\eta_i + [r_i]_{N_2}^{P_j} - 2\eta_i[r_i]_{N_2}^{P_j}$
   (c) If $\zeta = 0$,
   (d) $[\gamma_i]_{N_2}^{P_j} \leftarrow \tau_i\left(1 + (j-1)\eta_i - [r_i]_{N_2}^{P_j} + \sum_{k=i+1}^{\ell-1}[e_k]_{N_2}^{P_j}\right)$
   (e) Otherwise,
   (f) $[\gamma_i]_{N_2}^{P_j} \leftarrow \tau_i\left(1 + [r_i]_{N_2}^{P_j} - (j-1)\eta_i + \sum_{k=i+1}^{\ell-1}[e_k]_{N_2}^{P_j}\right)$
   (g) $[\hat{\gamma}_i]_{N_2}^{P_j} \leftarrow [\gamma_{i+\delta}]_{N_2}^{P_j}$
   (h) Send $[\hat{\gamma}_i]_{N_2}^{P_j}$ and η_0 to P_3

5: P_3
   (a) $\hat{\gamma}_i \leftarrow [\hat{\gamma}_i]_{N_2}^{P_1} + [\hat{\gamma}_i]_{N_2}^{P_2}$
   (b) If $\forall i, \hat{\gamma}_i \neq 0$, then $f \leftarrow \eta_0 \oplus r_0$; otherwise, $f \leftarrow 1 - \eta_0 \oplus r_0$
   (c) Generate $[f]_{N_1}^{P_j}$
   (d) Send $[f]_{N_1}^{P_j}$ to P_j

6: P_j $(j \in \{1, 2\})$
   (a) If $\zeta = 1$, $[c_0]_{N_1}^{P_j} \leftarrow (j-1) - [f]_{N_1}^{P_j}$

3.4.1 Security Analysis

Claim 1. *Protocol 8 is secure in the ($\mathcal{F}_{open}$)-hybrid model in presence of a semi-honest adversary.*

Proof. Since, the parties are under the semi-honest model; therefore, we only need to guarantee correctness and privacy. First, we start by proving the correctness of protocol 8 as follows:

- Based on the comparison reduction, we calculate $a - b$, then multiply the difference by 2. The result is held in c which is disguised later using the random value r. The result of the later step is represented by η. At the end of steps 2 and 3, η is revealed to party P_1 and P_2.

- Based on step 4, P_1 and P_2 calculate the xor between η_i and $[r_i]$, the result is assigned to the vector e. Then both parties calculate the vector γ based on equation 3.5. The resulted randomized vector $\hat{\gamma}$ is sent to P_3.

- P_3 adds up the shares of $\hat{\gamma}$ to reveal the values in each entry. Then check:

$$f = \begin{cases} \eta_0 \oplus r_0 & if\ \forall i, \hat{\gamma_i} \neq 0 \\ 1 - \eta_0 \oplus r_0 & otherwise \end{cases}$$

- Steps 5: The parties locally derive the shares of $[c_0]$ which holds the final comparison result, by de-randomizing $[f]$ such that

$$c_0 = \begin{cases} (j-1) - f & if\ \zeta = 1, where\ j \in \{1, 2\} \\ f & otherwise \end{cases}$$

Next, we prove the security in which we only need to guarantee privacy. This can be simply achieved by showing that every intermediate result which needs to be sent is a uniformly random value.

- In steps 2 and 3, although party P_1 and P_2 reveal η, it does not leak any information about a and b because r is not known to any of them and randomly chosen by P_3.

- In step 4, party P_1 and P_2 carry a local xor and then calculate γ vector based on equation 3.5. Note, that γ vector is masked using a random vector τ. Furthermore, the index is randomized using δ. Thus, party P_3 will receive a uniformly random vector with randomized indexes. Therefore, this step reveals nothing about a and b.

- At the end of the protocol P_1 and P_2 receive a fresh random value f which then can be locally de-randomized to get the final comparison result.

□

3.4.2 Complexity Analysis

The communication complexity is computed based on the number of bits sent or received.

- Step 1: $2\ell + 2\ell \log(\ell + 1)$
- Step 2: $\ell + \ell \log(\ell + 1) + \log \ell + 1$
- Step 3: ℓ
- Step 4: $2\ell \log(\ell + 1) + 1$
- Step 5: 2ℓ

The total number of bits is $6\ell + 5\ell \log(\ell + 1) + \log \ell + 2$. The round complexity is derived based on the number of sequence steps. According to Protocol 8, steps 2 and 3 can be performed in parallel. Therefore, the number of rounds is five. In addition, step 1 can be done in the offline phase, so alternatively, the protocol has one round for pre-processing and four rounds for online computation. Table 3.1 shows the complexity analysis comparing to the state of the art Wagh et al. secure comparison protocol [118].

Table 3.1: Complexity analysis

Protocol	Message complexity (bits)	Rounds
Protocol 8	$6\ell + 5\ell \log(\ell + 1) + \log \ell + 2$	5
Wagh et al. [118]	$5\ell^2 + 2\ell + \log(\ell) + 2$	4

3.5 Secure Comparison with Collision up to $n - 1$

In this section, we are ready to present an efficient design of secure comparison protocol against semi-honest adversaries. The protocol is based on the solutions with our proposed practical improvement. In the next chapter, we will show how to transform this protocol into a secure one in the malicious majority setting. Note that our proposed transformation technique works for any secure comparison protocol. The one presented in this section is very efficient in practice due smaller share size and less number of round. We will discuss the changes we made to make the protocol more efficient in practice. To begin with, we previously present a comparison reduction and two sub-protocols, Gen_Bitwise_Shares and Not_Zero, used to implement the comparison protocol.

Based on the two sub-protocols and the core ideas presented in Section 3.2 and Equation 3.3, we are ready to construct the comparison protocol that has a very good practical efficiency. The key steps are presented in Protocol 9.

- Step 1: The parties execute the Gen_Bitwise_Shares protocol is produce a randomly shared value $r \in \mathbb{Z}_p$ and its bitwise shares. This r will be used as a mask for the subsequent computations.

- Steps 2-4: According to the comparison reduction, first we compute the difference between a and b, and multiply the difference by 2. The result is stored in c which is masked later by adding the random value r. The sum is represented by η. At the end of step 4, η is revealed to every party. All these computations are performed on the shares. Although every party learns η, it does not leak any

information about a and b because r is not known to any party and randomly chosen.

- Step 5: Increment η by 1, and the result is denoted by $\hat{\eta}$. The subsequent computations are performed on $\hat{\eta}$ which is necessary to match the logic given in Equation 3.2. The explanation of step 12 given below include additional discussion regarding this issue.

- Steps 6-8: These steps closely follow the logic given in Equation 3.3 with slight modification. Step 6 computes bitwise XOR of $\hat{\eta}$ and r. The result is stored in e'_i. Since the shares are additive, a designated party (e.g., P_1), performs slightly different computation from the other parties so that the summation of shares of $[e'_i]$ gives $\hat{\eta}_i \oplus r_i$. Suppose i^* is the most significant bit location where $\hat{\eta} \neq r$. Steps 7 and 8 transform e' into $\hat{e}$, such that

  - $\hat{e}_i = 0$ for $i^* < i < \ell$, and
  - $\hat{e}_i \neq 0$ for $0 \leq i \leq i^*$

- Step 9: The parties execute the Not_Zero protocol and the result is stored in f where

$$f_i = \begin{cases} 0 & \text{for } i^* < i < \ell \\ 1 & \text{for } 0 \leq i \leq i^* \end{cases}$$

- Steps 10-11: The parties locally derive the shares of $[g_i]$, such that

$$g_i = \begin{cases} 0 & \text{for } i \neq i^* \wedge 0 \leq i < \ell \\ 1 & \text{for } i = i^* \end{cases}$$

- Step 12: The parties locally derive the shares of $[h]$:

$$h = \begin{cases} 1 & \text{if } \eta \geq r \\ 0 & \text{if } \eta < r \end{cases}$$

As discussed early, from step 6, the computations are performed on $\hat{\eta}$. If η were used, it would result

$$h = \begin{cases} 1 & \text{if } \eta > r \\ 0 & \text{if } \eta \leq r \end{cases}$$

Consequently, this would lead to an incorrect result for c_0 at the next step.

- Step 13: The parties derive the shares of $[c_0]$. Since $e'_0 = \hat{\eta}_0 \oplus r_0$ and $\hat{\eta} = \eta + 1$, we have $e_0 = \eta_0 \oplus r_0 = 1 - e'_0$ and c_0 can be derived accordingly:

$$c_0 = \begin{cases} e_0 = 1 - e'_0 & \text{if } h = 1 \\ 1 - e_0 = e'_0 & \text{if } h = 0 \end{cases}$$

This matches the result given in Equation 3.2. To compute $[1-h]$ (respectively $[1-e'_0]$), the parties perform the following with P_1 as a designated party:

  - $P_j (2 \leq j \leq n)$: $[1-h] \leftarrow p - [h]$
  - P_1: $[1-h] \leftarrow 1 - [h]$

Any party can serve the role of P_1, and $[1-e'_0]$ can be computed similarly.

3.5.1 Security and Complexity Analysis

Since we assume the adversary for protocol 9 is semi- honest, we do not need to guarantee the correctness of the protocol, and the protocol can also be aborted prematurely. Thus, its security proof is straightforward. Also, because the additive

Protocol 9 $SC([a],[b]) \rightarrow [c_0]$

Require: p is a prime, and $\ell = |p|$

1: $[r], [r_0], \ldots, [r_{\ell-1}] \leftarrow$ Gen_Bitwise_Shares(p)
2: $[c] \leftarrow 2([a] - [b])$
3: $[\eta] \leftarrow [c] + [r]$
4: $\mathcal{F}_{\text{open}}([\eta])$
5: $\hat{\eta} \leftarrow \eta + 1$
6: $P_j(2 \leq j \leq n) : [e'_i] \leftarrow [r_i] - 2\hat{\eta}_i[r_i]$
7: $P_1 : [e'_i] \leftarrow \hat{\eta}_i + [r_i] - 2\hat{\eta}_i[r_i]$
8: $[\hat{e}_{\ell-1}] \leftarrow [e'_{\ell-1}]$
9: $[\hat{e}_i] \leftarrow [e'_i] + [\hat{e}_{i+1}]$, for $i = \ell - 2$ down to 0
10: $[f_i] \leftarrow$ Not_Zero($[\hat{e}_i]$)
11: $[g_{\ell-1}] \leftarrow [f_{\ell-1}]$
12: $[g_i] \leftarrow [f_i] - [f_{i+1}]$, for $i = \ell - 2$ down to 0
13: $[h] \leftarrow \sum_{i=0}^{\ell-1} \hat{\eta}_i[g_i]$
14: $[c_0] \leftarrow [1 - h][e'_0] + [h][1 - e'_0]$

Functionality 10 $\mathcal{F}_{\text{sc}}([a],[b]) \rightarrow [\tau]$

Require: A prime number p that defines the share size

1: $\mathcal{F}_{\text{sc}}$ receives $\langle [a]^{\bar{\mathcal{A}}}, [b]^{\bar{\mathcal{A}}} \rangle$ from the honest parties, denoted by $P_{\bar{\mathcal{A}}}$. It also receives $\langle [a]^{\mathcal{A}}, [b]^{\mathcal{A}} \rangle$ from $\mathcal{A}$.
2: From these shares, $\mathcal{F}_{\text{sc}}$ derives a' and b' where $a' = a$ and $b' = b$ if the adversary did not modify the original shares.
3: Set $\tau = 0$ if $a' \geq b'$; otherwise, set $\tau = 1$.
4: Construct shares of $[\tau]^{\mathcal{A}}$ and $[\tau]^{\bar{\mathcal{A}}}$, and send $[\tau]^{\mathcal{A}}$ to $\mathcal{A}$.
5: $\mathcal{F}_{\text{sc}}$ waits for reply from $\mathcal{A}$:
   (a) If the reply is abort, send abort to $P_{\bar{\mathcal{A}}}$.
   (b) If the reply is continue, send $[\tau]^{\bar{\mathcal{A}}}$ to $P_{\bar{\mathcal{A}}}$.

shares provide privacy up to $n-1$ malicious parties and no values are reconstructed, protocol 9 ensures privacy.

Claim 2. *The protocol 9 securely implements $\mathcal{F}_{sc}$ with abort in the ($\mathcal{F}_{bitwise\_shares}$, $\mathcal{F}_{open}, \mathcal{F}_{not\_zero}, \mathcal{F}_{mult}$))-hybrid model in presence of a semi-honest adversary controlling at most $n-1$ parties.*

Proof. $\mathcal{S}_{\text{sc}}$ that accesses $\mathcal{F}_{\text{sc}}$ is constructed as follows:

- Receive $[a^*]^{\mathcal{A}}$ and $[b^*]^{\mathcal{A}}$ from the adversary $\mathcal{A}$.

- $\mathcal{S}_{\text{sc}}$ calls the simulator $\mathcal{S}_{bitwise\_shares(p)}$ of $\mathcal{F}_{\text{bitwise\_shares}}$, and let $[r^*]^{\mathcal{A}}, [r_0^*]^{\mathcal{A}}, \cdots, [r_{\ell-1}^*]^{\mathcal{A}}$ be the output from the simulator $\mathcal{S}_{bitwise\_shares}$.

- $\mathcal{S}_{\text{sc}}$ computes $[c^*]^{\mathcal{A}} = 2([a^*]^{\mathcal{A}} - [b^*]^{\mathcal{A}})$ and $[\eta^*]^{\mathcal{A}} = [c^*]^{\mathcal{A}} + [r^*]^{\mathcal{A}}$.

- $\mathcal{S}_{\text{sc}}$ calls the simulator $\mathcal{S}_{\text{open}}[\eta^*]^{\mathcal{A}}$ of $\mathcal{F}_{\text{open}}$, and let $[\eta]^{*^{\mathcal{A}}}$ be the output from $\mathcal{S}_{\text{open}}$. Set $\hat{\eta}^* = \eta^{*^{\mathcal{A}}} + 1$.

- $\mathcal{S}_{\text{sc}}$ simulates the rest of the values by randomly choosing the same number of values from $\mathbb{Z}_p$.

□

The first four steps simulate the exact interactions between the adversary and the real protocol. Since no values are reconstructed, the output is secretly shared, and we do not need to guarantee correctness, step 5 simulates the rest of the protocol by generating the same number of random values from $\mathbb{Z}_p$ as produced by the remaining steps of the protocol. Thus, the simulated view is computationally indistinguishable from the real view of the protocol.

The complexity of an SMC protocol is dominated by the number of secure multiplications ($\mathcal{F}mult$) being performed. Thus, a common practice in the literature is to use ($\mathcal{F}mult$) as the base unit to estimate the protocol complexity. Here we list the main steps of SC that require performing a non-constant number of $\mathcal{F}_{mult}$ calls.

- Step 1: the Bitwise Shares protocol makes call of Rand_b protocol which requires $\ell(n-1)$ multiplication invocations calls to $\mathcal{F}_{mult}$ where the n (indicating the number of parties) calls are incurred by the $Rand_b$ protocol.

- Step 9: the Not_Zero protocol makes approximately ℓ calls to $\mathcal{F}_{mult}$. Since Not Zero is performed ℓ times, the step requires l2 calls to $\mathcal{F}_{mult}$ in total.

Since n is generally much smaller than ℓ, we can simple say the complexity of SC is bounded by $O(\ell^2)$ $\mathcal{F}_{mult}$ operations.

3.6 Discussion

This section discusses different methods to generate random unknown values as well as various designs for the Not-Zero protocol as follows:

- Generating random unknown value associated with its random unknown bits can be done in three way as follows:
  - Protocol 3.3.2 presents a method to generate $[r], [r_0], \cdots, [r_\ell]$ such that $[r] \in \mathbb{Z}_p$. The protocol consumes 4ℓ multiplications in three rounds considering $[r_i]$ and $[s_i]$ to be generated in parallel.
  - The overhead of protocol 3.3.2 can be reduced by setting p to a Mersenne prime. Therefor, the check for $[r] < p$ which is utilized to be sure that $[r] \in \mathbb{Z}_p$ can be reduced to only bitwise equality test between the bits of p and $[r]$ which cost only one multiplication in one round. It is worth noting that Mersenne prime may not be applicable for FHE which is used to perform secure multiplication in the dishonest majority setting as explained in next chapter.
  - In a recent development, EdaBit method takes advantage of the mixed circuit to generate $[r] \in \mathbb{Z}_p$ and $[r_i] \in \mathbb{Z}_2$. EdaBit starts by allowing each party to generate a private random ℓ bits r_i and share them to other parties in the $\mathbb{Z}_2$ domain. Then each party calculates a private $r = \sum_{i=0}^{\ell} r_i 2^i$ and share it in $\mathbb{Z}_p$ domain. At this point the parties have the required shared to generate a global unknown r by adding the shares in both domains. It is worth mentioning that addition in $\mathbb{Z}_p$ can be done locally. However, addition in $\mathbb{Z}_2$ requires a binary n-input adder which costs $(\ell+\log n)\cdot(n-1)$ AND gates in $\ell \log n$ rounds. This last step leads to shares of the bits of r', without modular reduction. Further steps is required to guarantee that the generated r is of ℓ bits. This protocol becomes extremely expensive

in the presence of a dishonest majority adversary as it requires to check the consistency of each party input. Therefore, a cut-and-choose method is utilized.

- Not-Zero protocol can be designed in a constant round using symmetric Boolean function with complexity of 7 rounds and 17ℓ invocations [9]. The protocol can be implemented in two different methods in a logarithmic fashion [56]. The first method requires $\log(\ell)$ rounds and $\ell \log(\ell)/2$ invocations. While the other solution offers a trade-off between the rounds and the invocations complexity as it takes $2\log(\ell) - 1$ rounds and $2\ell - log(\ell) - 2$ invocations. It is worth noting that all the three solutions offer perfect security. On the other hand, the constant round solution requires extensive memory in the case of dishonest majority given a big number of comparisons.

3.7 Performance Evaluation

We empirically analyze the computational overhead of protocol 9 against the state of the art comparison protocol Rabbit [62]. Recall that protocol 9 requires only one random unknown value while Rabbit needs two random unknown values. In our test we utilize edaBit [96] to generate these random values. The most efficient edaBit implementation is included in the MP-SPDZ library [29]. Thus, we implemented our protocol and Rabbit using MP-SPDZ. Each data entry is secretly shared based on Shamir Secret Sharing SSS, Replicated Secret Sharing RSSS, or Additive Secret Sharing ASS scheme among 3 machines based on different adversary models. We used the Chameleon Cloud [119] to run our experiments, and each machine that represents one of the servers has the following specifications:

- Operating system: Ubuntu, Version = 18.04 LTS (Bionic Beaver).

- Machines hardware: x86_64 Cores: 12; Threads per core 2; Model: Intel®Xeon ®CPU E5-2670 v3 @ 2.30GHz, RAM 128GB.
- The ping between machines was 0.33 milliseconds.

To demonstrate the feasibility of the proposed protocol we implemented secure range query processing as an application. It is worth noting that the query processing requires to invoke two comparison protocols as well as two additional secure multiplications to get the final query result. The result is divided into two sections based on the adversary model as follows:

3.7.1 Semi-honest Model Results

Secure Comparison

We rigorously examine the runtime of the secure comparison protocol for different bit-lengths under the semi-honest setting. The data is secretly shared between three parties using Shamir Secret Sharing (SSS) and Replicated Secret Sharing (RSS) schemes. We take advantage of parallel processing as it is supported by MP-SPDZ and we set the number of threads to 8.

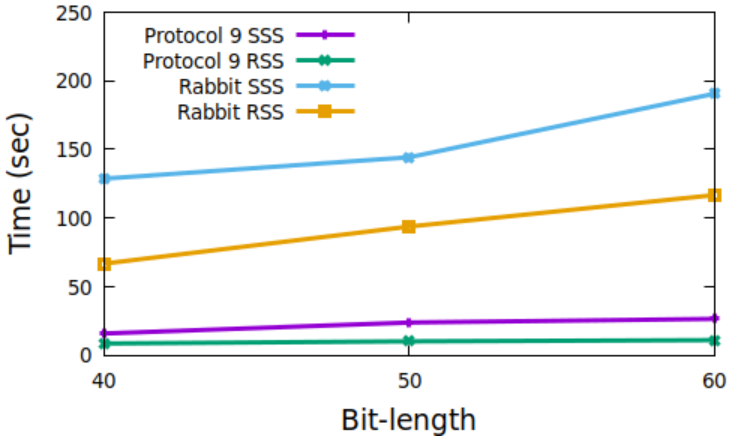

Figure 3.1: Runtime for different bit-lengths

Figure 3.1 (produced based on the data given in Table 3.2) illustrates the evaluation under 2M comparisons. Protocol 9 outperforms Rabbit by approximately 7 times and up to 10 times for SSS and RSS respectively.

We evaluate the amount of data communicated during protocol execution of 2M comparisons. We vary the bit-length = $\{40, 50, 60\}$ similar to the previous experiment. The number of parties is consisting of 3 parties, and the secret sharing schemes are set in the same manner as the last experiment. Figure 3.2 (produced based on numbers in Tables 3.3) depicts the data sent for different bit-lengths. The figure shows that protocol 9 requires to send 3.5 less data than Rabbit for both SSS and RSS schemes.

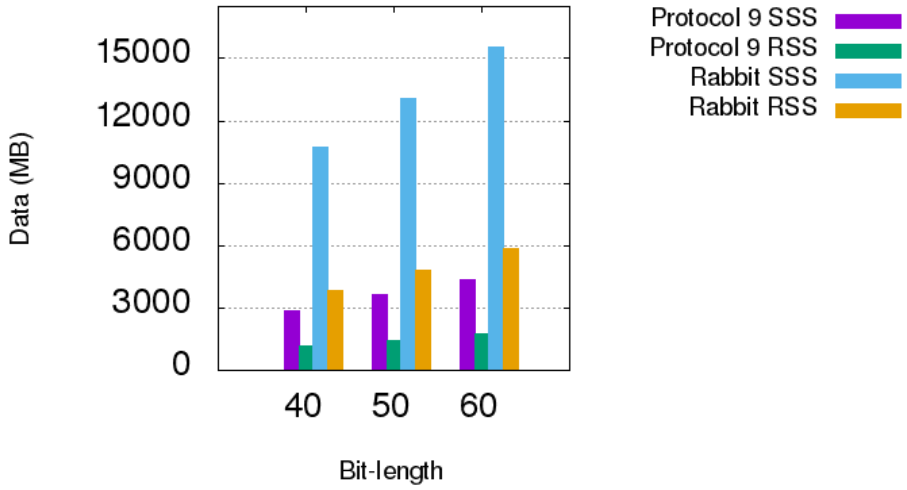

Figure 3.2: Data amount transferred for different bit-lengths

Table 3.2: Runtime for different bit-lengths

Bit-length	This SSS	This RSS	Rabbit SSS	Rabbit RSS
40	16.02	8.711	128.78	66.93
50	23.98	10.29	144.41	93.90
60	26.58	11.16	190.71	116.84

Table 3.3: Data transferred for different bit-lengths

Bit-length	This SSS	This RSS	Rabbit SSS	Rabbit RSS
40	2884	1167.99	10712	3805.96
50	3612	1444.55	13068	4809.08
60	4376	1739.11	15544	5872.2

Secure Range Query Processing

We thoroughly test the runtime of the secure range query processing protocol under the semi-honest setting. We set various bit lengths while fixing the number of queries to 2M. Using SSS or RSS schemes, the data is secretly shared between three parties. We utilize the parallel processing mode and we set the number of threads to 8. Figure

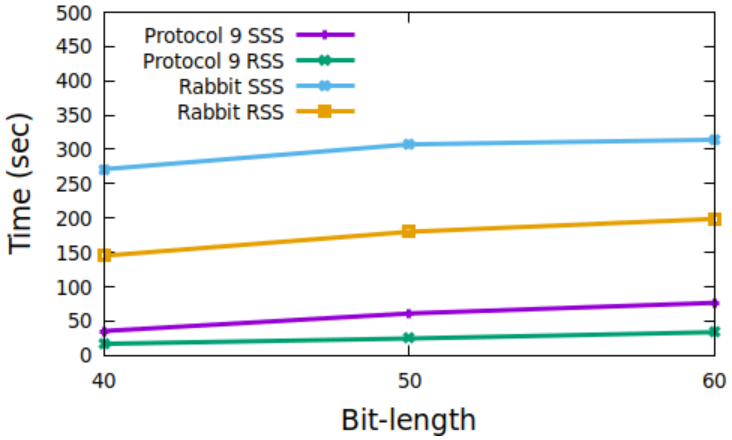

Figure 3.3: Runtime for different bit-length

3.3 (created using the data in Table 3.4) demonstrates the assessment for 2M queries. For SSS scheme, our protocol exceeds Rabbit by approximately 7 times. On the other hand, our protocol runs faster than Rabbit up to 9 times under RSSS scheme.

We examine the amount of data transferred between three parties during the protocol execution for 2M queries. We vary the bit-length $= \{40, 50, 60\}$ similar to the previous experiment. The secret sharing schemes are set in the same manner as the last experiment. Figure 3.4 (produced based on numbers in Tables 3.5) shows the data exchanged between the participants for different bit-lengths. The figure suggests that the query protocol based on protocol 9 requires to transfer 3.5 less data than

Rabbit for both SSS and RSS schemes.

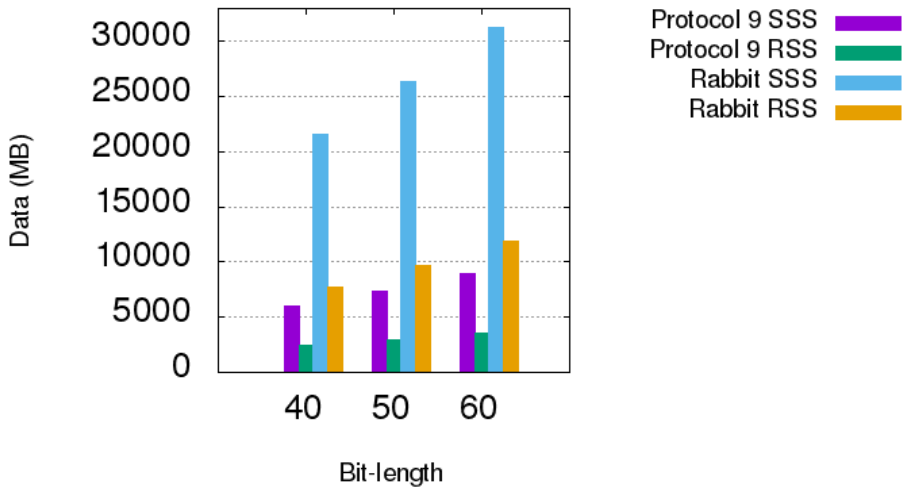

Figure 3.4: Data amount transferred for different bit-length

Table 3.4: Runtime for different bit-lengths

Bit-length	This SSS	This RSS	Rabbit SSS	Rabbit RSS
40	35.57	16.99	271.43	145.28
50	61.19	24.67	307.61	180.16
60	76.69	33.88	314.21	198.96

Table 3.5: Data amount transferred for different bit-length

Bit-length	This SSS	This RSS	Rabbit SSS	Rabbit RSS
40	5960	2431.98	21616	7707.91
50	7416	2985.1	26328	9714.15
60	8944	3574.22	31280	11840.4

3.7.2 Honest Majority Results

Secure Comparison

Under the honest majority model, we investigate the runtime of the secure comparison protocol for various bit-lengths. Using the Shamir Secret Sharing (SSS) and

Replicated Secret Sharing (RSS) schemes, the data is secretly shared among three parties. The number of threads is set to 8 following the previous experiments. Figure

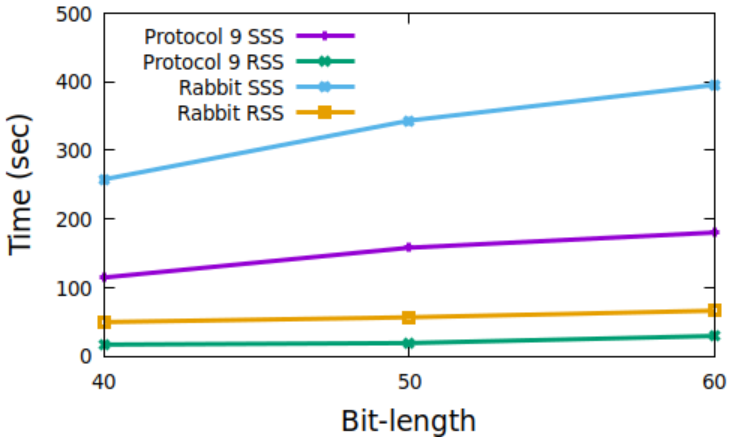

Figure 3.5: Runtime for different bit-length

3.5 (produced based on the data given in Table 3.6) illustrates the evaluation under 500K comparisons. Protocol 9 outperforms Rabbit by approximately 7 times and up to 10 times for SSS and RSS respectively.

We evaluate the amount of data communicated during protocol execution of 2M comparisons. We vary the bit-length = $\{40, 50, 60\}$ similar to the previous experiment. The number of parties is consisting of 3 parties, and the secret sharing schemes are set in the same manner as the last experiment. Figure 3.6 (produced based on numbers in Tables 3.7) depicts the data sent for different bit-lengths. The figure shows that protocol 9 requires to send 3.5 less data than Rabbit for both SSS and RSS schemes.

Table 3.6: Runtime for different bit-length

Bit-length	This SSS	This RSS	Rabbit SSS	Rabbit RSS
40	115.09	17.31	258.056	50.08
50	158.42	19.46	343.626	57.04
60	180.48	29.83	395.654	66.69

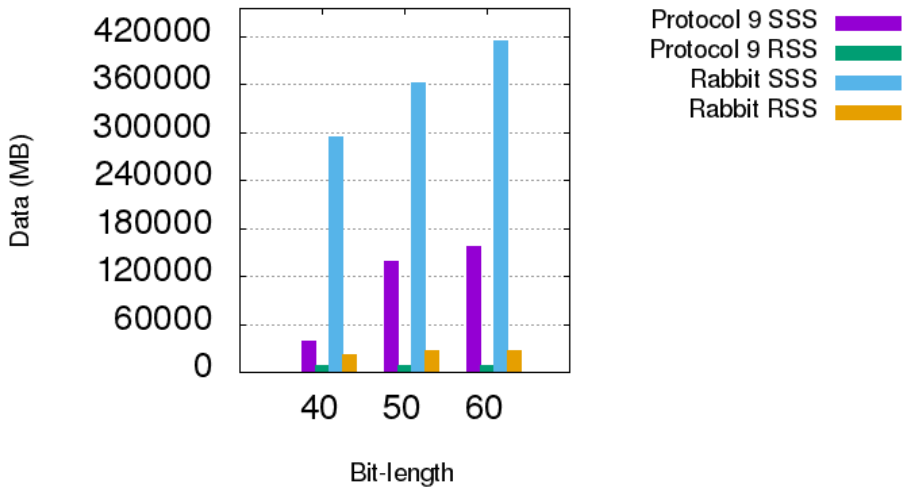

Figure 3.6: Data amount transferred for different bit-length

Table 3.7: Data amount transferred for different bit-length

Bit-length	This SSS	This RSS	Rabbit SSS	Rabbit RSS
40	38088	7656.23	295345	21769.7
50	139469	9004.91	362817	26464.1
60	157883	7927.22	415359	26802

Secure Query Processing

We test the runtime of the secure range query processing protocol under the honest majority setting. We set various bit lengths while fixing the number of queries to 2M. Using SSS or RSS schemes, the data is secretly shared between three parties.

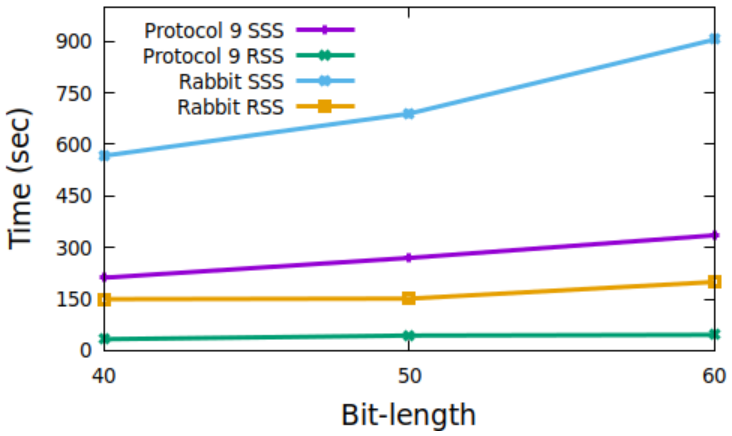

Figure 3.7: Runtime for different bit-length

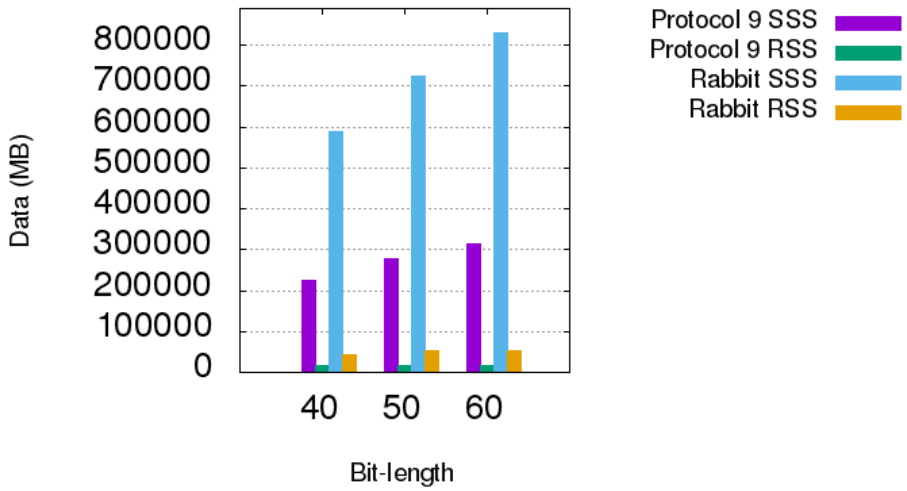

Figure 3.8: Data amount transferred for different bit-length

Table 3.8: Runtime for different bit-length

Bit-length	This SSS	This RSS	Rabbit SSS	Rabbit RSS
40	211.92	32.85	567.30	149.521
50	269.63	43.56	690.12	151.005
60	335.40	45.41	906.14	199.618

Table 3.9: Data amount transferred for different bit-length

Bit-length	This SSS	This RSS	Rabbit SSS	Rabbit RSS
40	223500	15296.7	589587	43548.6
50	277487	15883.1	724247	52909.7
60	314265	18016.3	829025	53604.8

Figure 3.7 (created using the data in Table 3.8) demonstrates that under SSS scheme, our protocol exceeds Rabbit by approximately 7 times. On the other hand, our protocol runs faster than Rabbit up to 9 times under RSSS scheme.

We examine the amount of data transferred between three parties during the protocol execution for 2M queries. We vary the bit-length $= \{40, 50, 60\}$ similar to the previous experiment. The secret sharing schemes are set in the same manner as the last experiment. Figure 3.8 (produced based on numbers in Tables 3.9) shows the data exchanged between the participants for different bit-lengths. The figure suggests

that our query protocol requires to transfer 3.5 less data than Rabbit for both SSS and RSS schemes.

Chapter 4

Generic Secure Comparison Compiler for Dishonest Majority

4.1 Introduction

Secure Multiparty Computation (SMC) allows a set of distrustful parties to jointly compute a function without revealing any information about their private input. In SMC, secure comparison (SC) serves as a fundamental operator in various data analytics. When the data considered under these applications contain sensitive information and are from multiple sources, privacy-preserving data analytics (PPDA) protocols may have to be adopted to protect the data and the outcomes. Although, SMC techniques provide very strong guarantee on personal privacy and data security, they are computationally expensive. For the last three decades, significant efforts have been devoted into developing efficient SMC primitives including SC.

When the majority (e.g., $n-1$ out of n) of the participating parties are malicious, to our knowledge, full malicious security cannot be achieved at least for SC protocols. The best can be done is to detect if any party behaved maliciously during protocol execution. These SC protocols [7–9, 117] are secure under the malicious model. How-

ever, their security is only guaranteed when the number of malicious parties is less than half of the total parties involved in the protocol execution. When the majority of parties are malicious (or dishonest majority), the current state of the art implementation of an SC to achieve malicious security is Rabbit [62] which adopts the edaBits protocol in the MP-SPDZ library [29]. MP-SPDZ provides implementations of different fundamental SMC techniques, such as secret sharing, oblivious transfer (OT) [120] and homomorphic encryption [71]. It allows mixing these techniques to achieve best efficiency.

In this chapter we propose a novel technique, termed as randomized replication, to develop a compiler that transforms semi-honestly secure SC protocols to be secure against malicious majority. More specifically, we consider the client-server computing model where clients outsource their data and analytics tasks to two or more independent servers. Most existing SMC solutions are applicable in the model. Our proposed compiler is also generic in that a newly developed and more efficient secure comparison protocol can be used without changing most of its code or structure. Since the clients are not involved in protocol execution, the servers are commonly referred to as the participating parties. As a result, an SC protocol under the client-server model may be formulated as:

$$SC(\langle P_i, [a]^{P_i}, [b]^{P_i} \rangle) \rightarrow \langle P_i, [\tau]^{P_i} \rangle$$

where a and b are actual values being compared, $[a]^{P_i}$ and $[b]^{P_i}$ are secret shares of a and b that are possessed by party P_i, and i varies from 1 to n. The comparison result is represented by τ, secretly shared among the n parties. Our contributions are as follows:

- We present a novel compiler that executes a semi-honest SC protocol κ times with randomized and replicated inputs plus end protocol verification, where κ

is a user chosen statistical security parameter.

- The κ parameter in our compiler is independent of the underling homomorphic encryption scheme. This leads to a small polynomial dimension N and a short ciphertext.
- We achieve higher statistical security than Rabbit up to 60-bit security based on the current SEAL library implementation. Since Rabbit is limited to 40-bit security in the MP-SPDZ library latest implementation.
- Our protocol achieve security in both covert and malicious models only by adjusting the actual value of κ.
- According to our empirical results and comparing to Rabbit, our solution has at least 5 times more efficient run-time and incurs at least 10 times less message complexity.

4.2 Building Blocks

4.2.1 Pairwise Secure Multiplication

Given two secret shared values $[x], [y]$, we aim to get their multiplication based on the pair wise multiplication observation as follows:

$$\begin{aligned} xy &= \left([x]^{P_1} + \cdots + [x]^{P_n}\right)\left([y]^{P_1} + \cdots + [y]^{P_n}\right) \\ &= \sum_{i=1}^{n} [x]^{P_i}[y]^{P_i} + \sum_{i=1}^{n}\sum_{j=1 \wedge i \neq j}^{n} [x]^{P_i}[y]^{P_j} \end{aligned}$$

The first summation is computed locally by each party, and the second summation utilizes the $\mathcal{F}_{\text{mult2}}$ functionality to produce secret shares of multiplication between each $\langle [x]^{P_i}, [y]^{P_j} \rangle$ pair. Then the share of $[xy]$ for each party can be derived by summing

all its local shares:

$$[xy]^{P_i} \leftarrow [x]^{P_i}[y]^{P_i} + \sum_{j=1 \wedge j \neq i}^{n} \mathcal{F}_{\text{mult2}}\left([x]^{P_i}, [y]^{P_j}\right)$$

$\mathcal{F}_{\text{mult}_2}(\langle P_i, \alpha\rangle, \langle P_j, \beta\rangle)$: the implementation of $\mathcal{F}_{\text{mult}_2}$ uses homomorphic encryption (HE) provided by the SEAL library [30]. The key steps are given in Protocol 11. The suitable HE parameters are chosen P_i according to the plaintext domain and security level specified by the system.

Protocol 11 $\text{Mult}_2(\langle P_i, \alpha\rangle, \langle P_j, \beta\rangle) \rightarrow \langle P_i, [\alpha\beta]^{P_i}\rangle, \langle P_j, [\alpha\beta]^{P_j}\rangle$

Require: p is prime of ℓ bits, Enc_{pk_i} is an encryption function with P_i's public key pk_i and Dec_{sk_i} is a decryption function with P_i's private key sk_i.

1: P_i: Compute $\text{Enc}_{pk_i}(\alpha)$ and send it to P_j

2: P_j:

(a) Select a random value $r \in \mathbb{Z}_p$

(b) Based on the homomorphic properties of HE, derive $c \leftarrow \text{Enc}_{pk_i}(\alpha\beta + r)$

(c) Set $[\alpha\beta]^{P_j} \leftarrow -r$ and send c to P_i

3: P_i: Set $[\alpha\beta]^{P_i} \leftarrow \text{Dec}_{sk_i}(c)$

Security Analysis

Like most SPDZ based protocols, we assume that the HE keys are generated by a trusted process, e.g., services provided by a public key infrastructure (PKI). As previously stated, these sub-protocols only need to guarantee privacy. As a result, the security proofs of these protocols are straightforward based on the real-ideal paradigm [14]. That is, for each protocol, we need to construct a simulator $\mathcal{S}$ that has access to the ideal functionality. To proof a protocol is secure, we just need to show the simulated execution image produced by $\mathcal{S}$ is computationally indistinguishable from that of the real execution.

Claim 3. *Protocol $Mult_2$ securely implements $\mathcal{F}_{mult2}$ when at most one party is malicious.*

Proof. This protocol is mostly the same as the ones presented in [12,45]. The protocol is secure as long as the parameters are chosen correctly. Referring to [45] for detailed analysis. The parameters used for our experimental analysis are discussed in Section 4.6. Note that the protocol does not guarantee correctness which is not necessary for constructing the proposed compiler. □

4.3 Secure Comparison in the Malicious Majority

In the previous protocols, a malicious party can modify the shares to produce invalid comparison results. For example, suppose P_1 is malicious and has shares $[a]^{P_1}$ and $[b]^{P_1}$. If the domain size l_s of the shares is at least twice as big as the domain size l_v of the actual values (i.e., a and b), P_1 can simply add $2^{l_v} - 1$ to its share of a. By doing so,the malicious party would have a very good chance of flipping the comparison outcome. The attack success rate is about $\frac{1}{2}$ analyzed below.

Except for aborting the protocol, a malicious behavior during protocol execution is equivalent to share modification. Therefore, we merely need to estimate the probability that by modifying the shares, how likely the comparison result of SC will change. Recall that the SC protocol returns 0 if $a \geq b$, and 1 otherwise. As discussed in Section 3.2, a and b are bounded by $\lfloor \frac{p}{2} \rfloor$. The following attack is feasible that may flip the comparison result:

- The adversary $\mathcal{A}$ modifies the shares $[a]^{\mathcal{A}}$ and $[b]^{\mathcal{A}}$ for each execution of $\mathcal{F}_{\text{sc}}$.

Let E be the event of flipping the comparison result by modifying the shares, and the probability of E can be estimated as follows assuming $a \geq b$ and $a < b$ are equally

likely to happen in practice:

$$\begin{aligned}\mathcal{P}(E) &= \mathcal{P}(a \geq b)\mathcal{P}(E|a \geq b) + \mathcal{P}(a < b)\mathcal{P}(E|a < b) \\ &= \frac{1}{2}\mathcal{P}(E|a \geq b) + \frac{1}{2}\mathcal{P}(E|a < b) \\ &= \frac{1}{2}\left(\mathcal{P}(b > a) + \mathcal{P}(b \leq a)\right) \\ &= \frac{1}{2}\end{aligned}$$

Such an attack can be easily carried out because the malicious party knows exactly which share to modify. In what follows, we will propose novel strategies to reduce the attack success rate to a negligible one.

Let κ be a statistical security parameter, and our goal is to detect malicious behaviors with probability bounded by $1 - \frac{1}{2^\kappa}$. To achieve this, the key idea in our design is for the participating parties to execute κ independent copies of the SC protocol with randomized input. By randomization, we mean the input shares $\langle [a], [b] \rangle$ are randomly permuted so that the malicious parties will not be able to consistently alter the shares across all κ copies. The following steps are needed to transform any semi-honest secure SC protocols into a secure one under the malicious model.

- Input commitment: The clients provide their inputs to the servers along with commitments to prevent input modification by the malicious servers.
- Randomized input replications: After the servers receive the shares, they randomize the ordering of the input to produce κ copies of the input pairs for subsequent κ independent SC computations.
- Output verification: Checking if all κ copies produce the same outputs.

4.3.1 Input Commitment

For the rest of this section, we use a triple $[a, \theta_a, \delta_a]$ to represent shares of the input value a, where δ_a is randomly chosen from $\mathbb{Z}_p$, and $\theta_a = a \cdot \delta_a$. The value θ_a serves as a message authentication code (MAC) for a. Such code is also used in [12]. To verify if the shares of a have been modified or not, the parties can perform the following verification steps:

- Collaboratively generate a random secretly shared value r from $\mathbb{Z}_p^+$. This can be done by randomly generate $[r]$ and $[s]$ from $\mathbb{Z}_p$. $[t] \leftarrow \mathcal{F}_{\text{mult}}([r], [s])$ and $\mathcal{F}_{\text{open}}([t])$. If $t = 0$, repeat these steps; otherwise, return $[r]$.
- Compute $[\omega] \leftarrow [r]([\theta_a] - [a][\delta_a])$.
- $\mathcal{F}_{\text{open}}([\omega])$ and examine:
  - If $\omega = 0$, verification passed.
  - If $\omega \neq 0$, verification failed.

As long as one party follows the steps, any modifications to the shares can be detect with probability $1 - \frac{1}{p}$. The goal of this verification is to make sure that when inputs are replicated, any malicious changes to the shares can be detected before executing the SC protocol. When p is small and not sufficient to achieve the desired security, we can increase either the size of p or the size of the authenticated shares. More details are discussed in Section 4.4.1.

4.3.2 Input Randomization and Replication

Input commitment only guarantees the inputs are valid; however, it cannot be used to verify if the parties followed the the prescribed steps during protocol execution. To be able to verify if the parties followed the protocol, the parties execute SC κ times

in parallel with randomized inputs. For example, we flip a coin κ times, the outcome of each coin flip is denoted by t_i for $1 \leq i \leq \kappa$. For succinctness, we use $[[a]]$ to represent $\langle [a], [\theta_a], [\delta_a] \rangle$, and adopt $\langle [[x]], [[y]], [t_i] \rangle$ or $\langle [[x]], [[y]] \rangle_{t_i}$ to represent the randomized input for the i-th SC execution based on t_i:

- $\langle [[x]], [[y]] \rangle_{t_i} = \langle [[a]], [[b]] \rangle$, if $t_i = 0$
- $\langle [[x]], [[y]] \rangle_{t_i} = \langle [[b]], [[a]] \rangle$, if $t_i = 1$

The randomization of the input shares has to be performed in an oblivious way so that the parties do not know which input shares are actually swapped. To achieve this, the parties perform the steps given in Protocol 12:

- Step 1: the parties generate a shared random bit $[t_i]$ for each input pair, and $[t_i]$ will also be used to de-randomize the output of the i-th SC execution.
- Step 2: from $[t_i]$, the parties generate a permutation matrix, denoted by $[M_{t_i}]$ for each input pair $\langle [[a]], [[b]] \rangle$. When $t_i = 0$, M_t is the 2-by-2 identity matrix. When $t_i = 1$, M_{t_i} is a transpose of the 2-by-2 identity matrix.
- Step 3: randomize each input pair by securely multiplying $\langle [[a]], [[b]] \rangle$ with $[M_{t_i}]$. Note that the secure matrix multiplication is applied to each pair of components of $[[a]]$ and $[[b]]$, i.e., $\langle [a], [b] \rangle$, $\langle [\theta_a], [\theta_b] \rangle$ and $\langle [\delta_a], [\delta_b] \rangle$. At the end, the protocol produces $\langle [[x]], [[y]], [t_i] \rangle$.

Protocol 12 Input_Rand$([[a]], [[b]]) \rightarrow \langle [[x]], [[y]], [t_i] \rangle$

Require: p is a prime defining the share domain.

1: $[t_i] \leftarrow \mathcal{F}_{\text{rand}_b}(p)$
2: $[M_{t_i}] \leftarrow \begin{Bmatrix} [1-t_i] & [t_i] \\ [t_i] & [1-t_i] \end{Bmatrix}$
3: $\langle [[x]], [[y]] \rangle_{t_i} \leftarrow \langle [[a]], [[b]] \rangle \times [M_{t_i}]$
4: return $\langle [[x]], [[y]], [t_i] \rangle$

At the step 3 of Protocol 12, to derive the shares of $[1 - t_i]$, a designated party, say P_1, sets its share $[1 - t_i]^{P_1} \leftarrow 1 - [t_i]^{P_1}$, and the other parties set their shares to

$[1-t_i]^{P_j} \leftarrow p - [t_i]^{P_j}$. The rest of the computations of the protocol can be carried out normal with standard secure additions and multiplications. The same set of permutation matrices can be used to permute the inputs for a number of SCs on different inputs as long as the comparison results are not leaked until the end of all SC executions.

Functionality 13 $\mathcal{F}_{\mathrm{sc}_m}\left(\left\langle [[a]]^{\mathcal{A}}, [[b]]^{\mathcal{A}}\right\rangle, \left\langle [[a]]^{\bar{\mathcal{A}}}, [[b]]^{\bar{\mathcal{A}}}\right\rangle\right) \rightarrow [[\tau]]^{\mathcal{A}}, [[\tau]]^{\bar{\mathcal{A}}}$

Require: $\left\langle [[a]]^{\mathcal{A}}, [[b]]^{\mathcal{A}}\right\rangle$ indicates the set of authenticated input shares controlled by the adversary $\mathcal{A}$, and $\left\langle [[a]]^{\bar{\mathcal{A}}}, [[b]]^{\bar{\mathcal{A}}}\right\rangle$ refers to the set of authenticated shares from honest parties. κ is the security parameter.

1: $\mathcal{F}_{\mathrm{sc}_m}$ receives $\left\langle [[a]]^{\bar{\mathcal{A}}}, [[b]]^{\bar{\mathcal{A}}}\right\rangle$ from the honest parties, denoted by $P_{\bar{\mathcal{A}}}$. It also receives $\left\langle [[a]]^{\mathcal{A}}, [[b]]^{\mathcal{A}}\right\rangle$ from $\mathcal{A}$.

2: From these shares, $\mathcal{F}_{\mathrm{sc}}$ derives $[a', \theta_{a'}, \delta_{a'}]$ and $[b', \theta_{b'}, \delta_{b'}]$:

   (a) If $\theta_{a'} \neq a' \cdot \delta_{a'}$ or $\theta_{b'} \neq b' \cdot \delta_{b'}$, send `abort` message to all parties.

3: Set $\tau = 0$ if $a' \geq b'$; otherwise, set $\tau = 1$.

4: Construct authenticated shares of $[[\tau]]^{\mathcal{A}}$ and $[[\tau]]^{\bar{\mathcal{A}}}$ according to the security parameter κ, and send $[[\tau]]^{\mathcal{A}}$ to $\mathcal{A}$.

5: $\mathcal{F}_{\mathrm{sc}_m}$ waits for reply from $\mathcal{A}$:

   (a) If the reply is `abort`, send `abort` to $P_{\bar{\mathcal{A}}}$.

   (b) If the reply is `continue`, send $[[\tau]]^{\bar{\mathcal{A}}}$ to $P_{\bar{\mathcal{A}}}$.

Protocol 14 $\mathrm{Open}_m([s]) \rightarrow s$

Require: $\mathcal{H}$ is secure commitment scheme that has κ-bit security, where κ is a statistically security parameter.

1: P_i broadcasts a commitment of its share $\mathcal{H}\left([s]^{P_i}\right)$ to the other parties.

2: After each party receives the commitments from all parities, P_i broadcasts a commitment of its share $[s]^{P_i}$.

3: If the received shares cannot be verified, abort the protocol

4: Reconstruct s from the shares.

Randomizing Inputs with Multiple Components

To permute a pair of random triples, we could apply the same permutation matrix on the corresponding components of each triple. Alternatively, we could first pact the

Protocol 15 Verify $([[a]])$

Require: $[[a]] \equiv \langle [a], [\theta_a], [\delta_a] \rangle$

1: $[t] \leftarrow \mathcal{F}_{\text{mult}}([a], [\delta_a])$
2: $[s] \leftarrow [t] - [\theta_a]$
3: Return $\mathcal{F}_{\text{verify\_zero}}([s])$

Protocol 16 Verify_Zero $([s])$

1: $[r_1] \leftarrow \mathcal{F}_{\text{rand}}(p)$ and $[r_2] \leftarrow \mathcal{F}_{\text{rand}}(p)$
2: $[t] \leftarrow \mathcal{F}_{\text{mult}}([r_1], [r_2])$
3: $t \leftarrow \mathcal{F}_{\text{open}_m}([t])$
4: `Abort` the protocol if $\mathcal{F}_{\text{open}_m}([t])$ aborts.
5: If $t = 0$, repeat the previous steps.
6: $[t'] \leftarrow \mathcal{F}_{\text{mult}}([r_1], [s])$
7: $t \leftarrow \mathcal{F}_{\text{open}_m}([t'])$
8: `Abort` the protocol if $\mathcal{F}_{\text{open}_m}([t])$ aborts.
9: If $t' = 0$, return `true`; otherwise, return `false`.

three components of a triple into a bigger share and generate a permutation matrix in a larger field to contain the bigger share. After multiplying the pair of the bigger shares with the permutation matrix, we can unpact the bigger shares to produce a permuted pair of triples.

4.3.3 Output Verification

The modified comparison will be performed κ times. After that we need to verify the consistency of these results. Let $[z_1], \ldots, [z_\kappa]$ be the results running the protocol κ times on randomized inputs. Before verifying the consistency of the results, we need to de-randomize them based on the t_i values used to randomize the input shares. However, we cannot simple de-randomize the results since the probability that the adversary alters the de-randomized results without being detected is no longer negligible. For instance, if the results are 0, then the adversary could add one to the share it controls across all κ results. This causes the comparison result flipped, and it is not guarantee to detect such malicious behaviors. As a solution, the parties need to authenticate the shares of z_i and t_i to produce $[[z_i]]$ and $[[t_i]]$ before de-randomization.

Share Authentication and Result De-randomization

The parties commit or authenticate their shares of the results before de-randomization. Let $[[z_1]], \ldots, [[z_\kappa]]$ be the authenticated shares of $z_1, \ldots, z_\kappa$, and $\tau_1, \ldots, \tau_\kappa$ be the de-randomized results. The parties can derive $[[\tau_i]]$ from $[[z_i]]$ and $[[t_i]]$ (see Section 4.4 for the details). To verify if the τ_i values are consistent, the parties perform the following steps:

- Derive $[[\tau]] \equiv \langle [\tau], [\theta_\tau], [\delta_\tau] \rangle$, where
  - $[\tau] \leftarrow \sum_{i=1}^{\kappa} [\tau_i]$
  - $[\theta_\tau] \leftarrow \sum_{i=1}^{\kappa} \kappa\, [\theta_{\tau_i}]$
  - $[\delta_\tau] \leftarrow \sum_{i=1}^{\kappa} [\delta_{\tau_i}]$
- Call $\mathcal{F}_{\text{verfy}}([[\tau]])$: If verification fails, we conclude malicious behavior occurred during protocol execution.

To see why the above verification works, we need to examine closely how the shares are derived. The security guarantee of the verification procedure is formalized in the next section.

4.4 The SC_m Protocol

Based on the proposed transformation techniques, we are ready to construct the SC_m protocol whose key steps are present in Protocol 17. In what follows, we discuss its step and the intuition behind the proposed design choices. Detailed security analysis is presented in Section 4.4.1. In the current version of the protocol, we assume that p is large enough to achieve κ-bit security. $\kappa = 40$ is sufficient for most applications. For efficiency considerations, we want to keep the size of p just adequate to accommodate the input values. Thus, we will later propose a strategy to achieve κ-bit security

Protocol 17 $\mathrm{SC}_m\left(\langle [[a]], [[b]]\rangle\right) \rightarrow [[c_0]]$

Require:

1: (a) p is a prime that defines the domain of the shares, and sufficiently large to provide κ-bit security.

(b) κ is a parameter indicating κ-bit statistical security.

(c) Index i is bounded by κ, i.e., $1 \leq i \leq \kappa$.

2: Input randomization and replication:

(a) $\langle [[x]], [[y]], [t_i]\rangle \leftarrow \mathcal{F}_{\text{input\_rand}}([[a]], [[b]])$

3: Input verification:

(a) For each $\langle [[x]], [[y]], [t_i]\rangle$, call $\mathcal{F}_{\text{verify}}([[x]])$ and $\mathcal{F}_{\text{verify}}([[y]])$

(b) The protocol aborts if any verification aborted or failed.

4: Executing κ secure comparison in parallel and generate authenticated shares for each result:

(a) $[z_i] \leftarrow \mathcal{F}_{\text{sc}}(\langle [x], [y]\rangle_{t_i})$

(b) $[\delta_{z_i}] \leftarrow \mathcal{F}_{\text{rand}}(p)$ and $[\delta_{t_i}] \leftarrow \mathcal{F}_{\text{rand}}(p)$

(c) $[\theta_{z_i}] \leftarrow \mathcal{F}_{\text{mult}}([z_i], [\delta_{z_i}])$ and $[\theta_{t_i}] \leftarrow \mathcal{F}_{\text{mult}}([t_i], [\delta_{t_i}])$

(d) $[[z_i]] \leftarrow \langle [z_i], [\theta_{z_i}], [\delta_{z_i}]\rangle$

(e) $[[t_i]] \leftarrow \langle [t_i], [\theta_{t_i}], [\delta_{t_i}]\rangle$

5: De-randomize the results and derive the authenticated shares of de-randomized results:

(a) $[\tau_i] \leftarrow [t_i] + [z_i] - 2 \times \mathcal{F}_{\text{mult}}([t_i], [z_i])$

(b) $[\delta_{\tau_i}] \leftarrow \mathcal{F}_{\text{mult}}([\delta_{t_i}], [\delta_{z_i}])$

(c) $[\theta_{\tau_i}] \leftarrow \mathcal{F}_{\text{mult}}([\theta_{t_i}], [\delta_{z_i}]) + \mathcal{F}_{\text{mult}}([\delta_{t_i}], [\theta_{z_i}]) - 2 \times \mathcal{F}_{\text{mult}}([\theta_{t_i}], [\theta_{z_i}])$

(d) $[[\tau_i]] \leftarrow \langle [\tau_i], [\theta_{\tau_i}], [\delta_{\tau_i}]\rangle$

6: Verifying the result:

(a) $[\tau] \leftarrow \sum_{i=1}^{\kappa} [\tau_i]$, $[\theta_\tau] \leftarrow \sum_{i=1}^{\kappa} \kappa\, [\theta_{\tau_i}]$, and $[\delta_\tau] \leftarrow \sum_{i=1}^{\kappa} [\delta_{\tau_i}]$

(b) Call $\mathcal{F}_{\text{verfy}}([[\tau]])$, and the protocol aborts if any verification returns either abort or fail.

7: Derive the final authenticated result:

(a) $[c_0] \leftarrow \kappa^{-1}\, [\tau]$ and $[\theta_{c_0}] \leftarrow \kappa^{-1}\, [\theta_\tau]$

(b) $[[c_0]] \leftarrow \langle [c_0], [\theta_{c_0}], [\delta_{c_0}]\rangle$ where $[\delta_{c_0}] \leftarrow [\delta_\tau]$

without increasing the size of p which leads to more efficient protocol implementations for small input domains.

The protocol takes authenticated shares from the clients or dealers. In real applications, a and b are generally belong to different clients. The shares are distributed to the servers or parties who perform the secure computations.

- **Step 1**: the original input shares are randomized and replicated κ times. Each time, a shared random bit t_i is generated to produce randomized input shares $[[x]]$ and $[[y]]$. Note that those are authenticated shares, but the shares of t_i are not. In the later steps, the protocol will generate authenticated shares for t_i. It is possible to generate the authenticated shares for t_i at this step.

- **Step 2**: Verify the integrity of the shares. The protocol aborts if any verification failed or aborted. This step captures if any malicious parties modified their shares during input randomization process.

- **Step 3**: For each randomized input pairs, call the $\mathcal{F}_{\text{sc}}$ functionality, and the randomized comparison result is secretly shared and stored in $[z_i]$. Next, the authenticated shares for both z_i and t_i are produced.

- **Step 4**: De-randomize the result to obtain the actual result τ_i for each comparison. Step 4(a) performs a secure xor of z_i and t_i; that is, $[\tau_i] = [z_i \oplus t_i]$. Then based on the authenticated shares $[[z_i]]$ and $[[t_i]]$, the protocol derives the authenticated shares of τ_i. First, $[\delta_{\tau_i}] = [\delta_{z_i}\delta_{t_i}]$ is derived by a secure multiplication of $[\delta_{z_i}]$ and $[\delta_{t_i}]$. Then the protocol computes $[\theta_{\tau_i}]$:

$$\begin{aligned}
[\theta_{\tau_i}] &= [\theta_{t_i}][\delta_{z_i}] + [\theta_{z_i}][\delta_{t_i}] - 2[\theta_{t_i}][\theta_{z_i}] \\
&= [t_i\delta_{t_i}][\delta_{z_i}] + [z_i\delta_{z_i}][\delta_{t_i}] - 2[t_i\delta_{t_i}][z_i\delta_{z_i}] \\
&= [t_i\delta_{t_i}\delta_{z_i} + z_i\delta_{z_i}\delta_{t_i} - 2t_iz_i\delta_{t_i}\delta_{z_i}] \\
&= [(t_i + z_i - 2t_iz_i)\,\delta_{t_i}\delta_{z_i}] \\
&= [\tau_i\delta_{\tau_i}]
\end{aligned}$$

- **Step 5**: verify the comparison result based on the strategy discussed in Section 4.3.3. The protocol aborts if the verification aborted or failed.

- **Step 6**: derive the final authenticated comparison result. κ^{-1} indicates the multiplicative inverse of κ in $\mathbb{Z}_p$. If the verification passed at Step 5, τ is either 0 or κ and θ_τ is either 0 or $\kappa\delta_\tau$. That is:

$$[[\tau]] = \begin{cases} \langle [0], [0], [\delta_\tau] \rangle & \text{if } \tau = 0 \\ \langle [\kappa], [\kappa\delta_\tau], [\delta_\tau] \rangle & \text{if } \tau = \kappa \end{cases}$$

  Thus, after multiplying κ^{-1} with $[\tau]$ and $[\theta_\tau]$, c_0 is set to 0 or 1, and $[[c_0]]$ is returned as

$$[[c_0]] = \begin{cases} \langle [0], [0], [\delta_\tau] \rangle & \text{if } c_0 = 0 \\ \langle [1], [\delta_\tau], [\delta_\tau] \rangle & \text{if } c_0 = 1 \end{cases}$$

At the end of the protocol, each party sends its shares of $[[c_0]]$ to the client whose reconstructs c_0, θ_{c_0} and δ_{c_0}. The client accepts the result if $\theta_{c_0} = c_0\delta_{c_0}$. If SC_m is adopted as a sub-protocol, then $[[c_0]]$ can be directly used for the subsequent secure computations.

The protocol can be simplified by requiring the client to perform some extra computations. At Step 3, the shares of κ pairs of $[[z_i]]$ and $[[t_i]]$ can be returned to the client who then verifies the authenticated shares and de-randomize the comparison results. The client accepts the result if all verifications passed and all κ de-randomized results are the same. Alternatively, if SC_m is a subroutine, the execution could end at Step 4 where the $[[\tau_i]]$ values would be used for the subsequent computations. We will provide more details about this alternative in the range query application shortly.

4.4.1 Security Analysis

In this section, we analyze the security of SC_m using the real-ideal paradigm. First, we prove the following claim related to the output verification step of SC_m.

Claim 4. *If the parties follow the SC_m protocol, $\mathcal{F}_{verfy}([[\tau]])$ will succeed. On the other hand, if any shares are modified after $[[z_i]]$ and $[[t_i]]$ are generated, the verification will fail with probability*

$$\min\left(1-\frac{1}{p}, 1-\frac{1}{2^\kappa}\right)$$

Proof. Suppose the parties follow the protocol to derive $[[\tau]]$ correctly, then all τ_i values are either 0 or 1, and we have

$$\theta_\tau = \sum_{i=1}^{\kappa} \kappa\theta_{\tau_i} = \sum_{i=1}^{\kappa} \kappa\tau_i\delta_{\tau_i} = \kappa\tau_i\delta_\tau \tag{4.1}$$

If $\tau_i = 0$, then $\tau = 0$ and $\theta_\tau = 0$. If $\tau_i = 1$, then $\tau = \kappa$ and $\theta_\tau = \kappa\delta_\tau$. In either case, θ_τ is derived correctly. As a consequence, the verification will go through successfully.

Next we analyze the probability that the verification passes when any shares could be modified by the adversary $\mathcal{A}$. During the verification process, the parties compute the following values:

$$\tau\delta_\tau = \hat{\tau}\sum_{i=1}^{\kappa}\hat{\delta}_{\tau_i} = \hat{\tau}\sum_{i=1}^{\kappa}\hat{\delta}_{t_i}\hat{\delta}_{z_i} \tag{4.2}$$

$$\theta_\tau = \hat{\kappa}\sum_{i=1}^{\kappa}\left(\hat{t}_i + \hat{z}_i - 2\hat{t}_i\hat{z}_i\right)\hat{\delta}_{t_i}\hat{\delta}_{z_i} \tag{4.3}$$

Since the adversary could modify any values in either equation, let $\hat{\tau}$, $\hat{\kappa}$, $\hat{t}_i$, $\hat{z}_i$, $\hat{\delta}_{t_i}$ and $\hat{\delta}_{z_i}$ denote the values after their shares were modified by $\mathcal{A}$. However, the adversary had no control over what the actual values would be after modifying the shares of these values because t_i, z_i, δ_{t_i} and δ_{z_i} were randomly generated. As a result, if $\tau \neq 0$ or $\tau \neq \kappa$, the chance for the two equation to return the same value is $\frac{1}{p}$. In addition,

if z_i and t_i could be correctly predicted, the verification would pass as long as $\hat{\tau} = \hat{\kappa}$ in both equations. Nevertheless, since each pair of input was randomized, and t_i was randomly generated, the probability that the adversary can predict both values correctly is bounded by $\frac{1}{2^\kappa}$ because there are κ input pairs and t_i values.

Another scenario is that the adversary followed the protocol until the end step 4. The adversary may try to flip the result during step 5. It is easy to change τ from 0 to κ, and vice versa, but still making the equality $\tau\delta_\tau = \theta_\tau$ valid is difficult for the adversary due to the fact that δ_τ is random. The probability of the equality is valid after any modifications to the shares of τ, δ_τ, θ_τ, τ_i, δ_{τ_i} and θ_{τ_i} is bounded by $\frac{1}{p}$.

Combining these probabilities, the overall probability of passing the verification when τ is neither 0 or κ is $\max\left(\frac{1}{p}, \frac{1}{2^\kappa}\right)$ which leads to the failure probability given in the claim. □

Claim 5. *Protocol SC_m securely implements $\mathcal{F}_{sc_m}$ with abort in the $\left(\mathcal{F}_{input\_rand}, \mathcal{F}_{verify}, \mathcal{F}_{sc}, \mathcal{F}_{rand}, \mathcal{F}_{verify\_zero}, \mathcal{F}_{mult}\right)$-hybrid model in presence of a malicious adversary controlling at most $n-1$ parties.*

Proof. We prove the security of SC_m under the universally composable security model [121]. The main idea is to build a simulator that interacts with the ideal functionalities. If the protocol is secure, then the environment cannot distinguish a real execution of SC_m from an ideal execution between the simulator and the functionalities. The simulator $\mathcal{S}_{\mathrm{sc}_m}$ is constructed as follows:

- Receive $[[a]]^{\mathcal{A}}$ and $[[b]]^{\mathcal{A}}$ from the adversary $\mathcal{A}$.
- $\mathcal{S}_{\mathrm{sc}_m}$ calls the simulator $\mathcal{S}_{\mathrm{input\_rand}}\left([[a]]^{\mathcal{A}}, [[b]]^{\mathcal{A}}\right)$ of $\mathcal{F}_{\mathrm{input\_rand}}$, and let $\left\langle [[x^*]]^{\mathcal{A}}, [[y^*]]^{\mathcal{A}}, [t_i^*]^{\mathcal{A}} \right\rangle$ be the i-th output from $\mathcal{S}_{\mathrm{input\_rand}}$, where $1 \leq i \leq \kappa$.
- For each $\left\langle [[x^*]]^{\mathcal{A}}, [[y^*]]^{\mathcal{A}}, [t_i^*]^{\mathcal{A}} \right\rangle$, $\mathcal{S}_{\mathrm{sc}_m}$ calls the simulators $\mathcal{S}_{\mathrm{verify}}\left([[x^*]]^{\mathcal{A}}\right)$ and $\mathcal{S}_{\mathrm{verify}}\left([[y^*]]^{\mathcal{A}}\right)$ of $\mathcal{F}_{\mathrm{verify}}$. $\mathcal{S}_{\mathrm{sc}_m}$ outputs `abort` if any $\mathcal{S}_{\mathrm{verify}}$ returned `abort`.

- For each $\left\langle [[x^*]]^{\mathcal{A}}, [[y^*]]^{\mathcal{A}}, [t_i^*]^{\mathcal{A}} \right\rangle$ $\mathcal{S}_{\mathrm{sc}_m}$ calls the simulator $\mathcal{S}_{\mathrm{sc}}\left(\left\langle [[x^*]]^{\mathcal{A}}, [[y^*]]^{\mathcal{A}} \right\rangle_{t_i^*}\right)$ of $\mathcal{F}_{\mathrm{sc}}$, and let $[z_i^*]^{\mathcal{A}}$ be the output of each call to $\mathcal{S}_{\mathrm{sc}}$.

- $\mathcal{S}_{\mathrm{sc}_m}$ calls the simulator $\mathcal{S}_{\mathrm{rand}}(p)$ of $\mathcal{F}_{\mathrm{rand}}$ 2κ times to produce $\left[\delta_{z_i}^*\right]^{\mathcal{A}}$ and $\left[\delta_{t_i}^*\right]^{\mathcal{A}}$.

- $\mathcal{S}_{\mathrm{sc}_m}$ calls the simulator $\mathcal{S}_{\mathrm{mult}}\left([z_i^*]^{\mathcal{A}}, \left[\delta_{z_i}^*\right]^{\mathcal{A}}\right)$ and $\mathcal{S}_{\mathrm{mult}}\left([t_i^*]^{\mathcal{A}}, \left[\delta_{t_i}^*\right]^{\mathcal{A}}\right)$ of $\mathcal{F}_{\mathrm{mult}}$. Let $\left[\theta_{z_i}^*\right]^{\mathcal{A}}$ and $\left[\theta_{t_i}^*\right]^{\mathcal{A}}$ be the output of $\mathcal{S}_{\mathrm{mult}}$.

- $\mathcal{S}_{\mathrm{sc}_m}$ calls $\mathcal{S}_{\mathrm{mult}}\left([t_i^*]^{\mathcal{A}}, [z_i^*]^{\mathcal{A}}\right)$, $\mathcal{S}_{\mathrm{mult}}\left(\left[\delta_{t_i}^*\right]^{\mathcal{A}}, \left[\delta_{z_i}^*\right]^{\mathcal{A}}\right)$, $\mathcal{S}_{\mathrm{mult}}\left(\left[\theta_{t_i}^*\right]^{\mathcal{A}}, \left[\delta_{z_i}^*\right]^{\mathcal{A}}\right)$, $\mathcal{S}_{\mathrm{mult}}\left(\left[\delta_{t_i}^*\right]^{\mathcal{A}}, \left[\theta_{z_i}^*\right]^{\mathcal{A}}\right)$, $\mathcal{S}_{\mathrm{mult}}\left(\left[\theta_{t_i}^*\right]^{\mathcal{A}}, \left[\theta_{z_i}^*\right]^{\mathcal{A}}\right)$, and $\mathcal{S}_{\mathrm{mult}}\left([t_i^*]^{\mathcal{A}}, \left[\delta_{t_i}^*\right]^{\mathcal{A}}\right)$. From the outputs, $\mathcal{S}_{\mathrm{sc}_m}$ derives $[[\tau_i^*]]^{\mathcal{A}}$ and $[[\tau^*]]^{\mathcal{A}}$.

- $\mathcal{S}_{\mathrm{sc}_m}$ calls $\mathcal{S}_{\mathrm{verify}}\left([[\tau^*]]^{\mathcal{A}}\right)$. $\mathcal{S}_{\mathrm{sc}_m}$ outputs `abort` if $\mathcal{S}_{\mathrm{verify}}$ returned `abort`.

- $\mathcal{S}_{\mathrm{sc}_m}$ sends $[[\tau^*]]^{\mathcal{A}}$ to $\mathcal{A}$, and outputs whatever $\mathcal{A}$ outputs.

It is straightforward to check that the simulated view produced by $\mathcal{S}_{\mathrm{sc}_m}$ is computationally indistinguishable from the real view. □

4.4.2 Complexity Analysis

In this section, we provide a detailed complexity analysis for both computation and communication. We first start by defining the criteria to our analysis as follows:

1. Multiplication invocations: to show the computation complexity we refer to the most expensive operation in the SMC paradigm which is the multiplication operation. Since Rabbit, edaBit, and daBit work in the mixed binary and arithmetic circuits then we will summarize two types of multiplication in $\mathbb{F}_2$ and $\mathbb{F}_p$ for binary and arithmetic circuits respectively. It is worth noting that our proposed solutions work in the $\mathbb{F}_p$.

2. Round complexity: it refers to the number of send/receive operations during a protocol execution. It is worth noting that independent rounds can be carried in parallel; thus, they are considered as one round.

3. Similar to other work, the complexity of sharing and revealing (open) is considered negligible and we only consider them toward the communication rounds.

$\mathbf{SC}_m$ Complexity

This section presents the detailed complexity analysis of protocol 17 as follows:

- Step 1: it requires $\kappa(n-1) + 12\kappa$ of the $\mathbb{F}_p$ multiplication in $\log n + 1$ rounds.
- Step 2: it takes 6 of the $\mathbb{F}_p$ multiplication in 6 rounds.
- Step 3-a: this step requires $(\ell n + 20\ell + 2)$ of the $\mathbb{F}_p$ multiplication in $21 + \log n$ rounds.
- Step 3-b: it needs 1 of the $\mathbb{F}_p$ multiplication in 1 round.
- Step 3-c,d: these steps require 4κ of the $\mathbb{F}_p$ multiplication in 2 rounds.
- Step 4: this step consists of four sub steps which need 4κ of the $\mathbb{F}_p$ multiplication in 3 round.
- Step 5-b: this step takes 3 of the $\mathbb{F}_p$ multiplication in 6 rounds.
- Step 6: it requires 1 of the $\mathbb{F}_p$ multiplication in 1 round.

Table 4.1 shows the total SC_m complexity. In the next section, we show the complexity of multiplication in both $\mathbb{F}_2$ and $\mathbb{F}_P$. Where $\mathbb{F}_2$ multiplication is done using TinyOT while $\mathbb{F}_p$ is done using HE.

Table 4.1: SC complexity

Step	Mult-F_p	Rounds
Total	$\kappa(n + \ell n + 20\ell + 21) + 11$	$41 + 2\log n$

Multiplication in $\mathbb{F}_2$

In this section, we consider the complexity reported in [51]. It is worth noting that the number of rounds was not reported in [51]. Therefore, we calculate the rounds based on Authenticate protocol in figure 6, COTE protocol in figure 19, and MACCHECK protocol in figure 16 in [51]. Table 4.2 shows the complexity of $\mathbb{F}_2$ multiplication.

Table 4.2: Complexity of $\mathbb{F}_2$ multiplication

Steps	(1-2)OT	COTe	Rounds
Total	$9n(n-1)$	$27n(n-1)$	13

Multiplication in $\mathbb{F}_p$

To analyze the complexity of arithmetic multiplication under $\mathbb{F}_p$, first we report the complexity of ZK-proof based on figure 1 in [13]. It is worth noting that during this analysis we do not consider the batching in the HE. We report the complexity using V which refer to the auxiliary ciphertext, κ the soundness of ZK proof; the complexity is as follows:

- Step Samp-3: this step is done locally by every P_i and it requires $2nV$ encryption operations.
- Step Samp-4: it takes 1 round to broadcast C.
- Step Comm-2: it requires nV encryption operations.
- Step Comm-3: this step needs two rounds to broadcast the commitments.
- Step Chall-1: it takes 1 round to generate the random challenge matrix W.

- Step Resp-2: it requires 1 round to broadcast the response.
- Step Verify: this step needs nV encryption operations.

The total complexity is shown in table 4.3. Note that Samp, Comm, Chall can be carried in parallel; thus, they require two round. Therefore, the total rounds become three.

Table 4.3: ZK-proof complexity

Steps	Enc	Rounds
Total	$n4V$	3

Next, we analyze the complexity of generating the triples based on figure 3 in [13] as follows:

- Init-1: this is local operation which takes n encryption operations.
- Init-2: it takes 1 rounds for commitments.
- Init-3: it requires $2n\kappa$ encryption operations and 2 rounds.
- Triples-2: this step needs $12nV$ encryption operations.
- Triples-3: it takes 2 rounds.
- Triples-4: this step requires $12nV$ encryption operations in addition to two rounds.
- Triples-6: it takes $4nV$ ciphertext multiplications as well as two rounds.
- Triples-7: this step requires $4nV$ decryption operations and 1 rounds.
- Triples-9: it takes $4nV$ encryption operations.
- Triples-10: it needs $12nV$ ciphertext multiplications.
- Triples-11: this step requires $12nV$ decryption operations and 1 rounds.

Since $V = (\kappa + 2)/\log_2(2N + 1)$ and $\log_2(2N + 1) \approx 16$, the total complexity of triple generation is illustrated in table 4.4; the table also summarizes the complexity of multiplication in this paper. Due to the fact that our randomized replication only requires to run the comparison κ times with the verification step at the end, this leads to efficient multiplication protocol with only one round. While TopGear protocol of the SPDZ compiler requires 11 rounds.

Table 4.4: Complexity of $\mathbb{F}_p$

Steps	Enc	Dec	Mult-ct	Rounds
This paper	$n\kappa$	$n\kappa$	$n\kappa$	1
Total	$n(2\kappa + (7\kappa + 14)/4 + 1)$	$n(\kappa + 2)$	$n(\kappa + 2)$	11

Rabbit Complexity

To analyze Rabbit complexity we are required to first figure out the complexity of edabit which invokes dabit. Therefore, we will start our analysis from dabit going up to Rabbit.

DaBit Complexity: The daBit starts by generating faulty daBits (not yet checked) then check them. The complexity of generating faulty daBit is based on figure 13 in [28] as follows:

- Step 1: this step requires one round.
- Step 2: it takes n of the $\mathbb{F}_p$ multiplications.

While the complexity of checking the faulty daBits is based on figure 16 in [28] as follows:

- Step 1-b: it requires 1 round.
- Step 1-c: it takes 1 round.

- Step 3: this step needs 1 of the $\mathbb{F}_p$ multiplication in 12 rounds.

Adding the complexity of faulty daBit and checking them leads to the total complexity in table 4.5.

Table 4.5: Total daBit complexity

	Mult-F_2	Mult-F_p	Rounds
Total	-	$n+1$	$11\log n + 14$

EdaBit Complexity: To generate edaBit, the parties first produce private edaBits, then check these private edabit for every party. Finally, the parties add the checked private edaBits to all the other parties' private edaBits, created in the same way, to obtain secret-shared global edaBits. We start the analysis by first analyzing the private edabit as follows:

- Generate/share ℓ bits: this step takes only one round.
- Generate/share r: it requires one round.
- Generate/share triples: this also needs one round.

It is worth noting that the steps of generating private daBits are independent and can be carried in one round. Next, we analyze the check procedure based on the Cut-and-Choose technique in figure 5 in [28] as follows:

- Step 2: it requires one round.
- Step 4-b: it also takes one round.
- 4-c: this step requires $n^2 2^{\kappa/(B-1)}B + n2^{\kappa/(B-1)}B$ of the $\mathbb{F}_p$ multiplications which takes $11\log n + 14$ round.
- 4-d: it requires 1 round.

The complexity of going from private daBits to global daBits based on figure 3 in [28] is as follows:

- private edabit: it takes 1 round.
- CutNChoose: this step takes $n^2 2^{\kappa/(B-1)}B + n2^{\kappa/(B-1)}B$ of the $\mathbb{F}_p$ multiplication in $17 + 11\log n$ rounds.
- Step 3: this step requires $(\ell + \log n) \cdot (n-1)$ of the $\mathbb{F}_2$ multiplication in $13\ell \log n$ rounds.
- Step 4: this step needs $n+1$ of the $\mathbb{F}_p$ multiplication in $11\log n + 14$ rounds.

Adding up the complexity of private edaBit, checking edaBit and global edaBit gives the total complexity as in table 4.6.

Table 4.6: Complexity of edaBit

Step	Mult-F_2	Mult-F_p	Rounds
Total	$n\ell + n\log n$	$n^2 2^{\kappa/(B-1)}B + n2^{\kappa/(B-1)}B + n + 1$	$32 + 13\ell\log n + 22\log n$

Rabbit Complexity: Now we are ready to present the complexity of Rabbit protocol based on figure 6 in [62] as follows:

- Two edaBits: this step requires $2n\ell + 2n\log n$ of $\mathbb{F}_2$ multiplication and $2n^2 2^{\kappa/(B-1)}B + 2n2^{\kappa/(B-1)}B + 2n + 2$ of $\mathbb{F}_p$ multiplication both in $32 + 13\ell\log n + 22\log n$ rounds.
- Step 2: it takes 1 round.
- Step 3-1,b: these two steps take 42ℓ of $\mathbb{F}_2$ multiplication in 234 rounds.
- Step 3-d: it requires ℓ of $\mathbb{F}_2$ multiplication in ℓ rounds.
- Step 3-e: it takes 21ℓ of $\mathbb{F}_2$ multiplication in 234 rounds.

- Step 4: this step requires $n+1$ of $\mathbb{F}_p$ multiplication in $17+11\log n$ rounds.

The total complexity of Rabbit comparing to the complexity of SC_m is illustrated in table 4.7.

Table 4.7: Rabbit complexity

Step	Mult-F_2	Mult-F_p	Rounds
SC_m	-	$\kappa(n+\ell n+20\ell+21)+11$	$41+2\log n$
Rabbit	$2n\ell+64\ell+2n\log n$	$2n^2 2^{\kappa/(B-1)}B+2n2^{\kappa/(B-1)}B+3n+3$	$518+13\ell\log n+33\log n+\ell$

4.5 Discussion

In this section, we provide a number of points on our work. We elaborate on dealing with smaller share sizes as well as removing the slack in the comparison protocol, and close the discussion with statistical security argument.

Dealing with Smaller Share Sizes

This section addresses issues related to the share size, especially for authenticated shares. For practical consideration, we limit the size of p to be as small as the input domain, e.g., 32-bit integer. In this case, the authenticated shares only offer 32-bit security at most. If $\kappa = 50$ (sufficient for most existing applications), we could increase the size of p to match κ-bit security. However, if 32-bit domain is sufficient for the actual input data, κ-bit shares will increase not only the computation complexity but also incur large communication cost. Here we proposed an efficient solution to ensure the security guarantee of the authenticated shares matches the overall security of the SC protocol. Our idea is to use different share sizes for the authenticated shares.

Recall that $[[a]]$ represents $\langle [a], [\theta_a], [\delta_a] \rangle$, and each component is generated with the same p. To increase the security level, we can generate $[\theta_a]$ and $[\delta_a]$ with large share size. However, we cannot simply select a large prime q and perform computations in

$\mathbb{Z}_q$ since the computations are not compatible between two unrelated fields. Instead, let $\mathbb{K}$ denote an extension field of $\mathbb{Z}_p$ that is sufficiently large to achieve κ-bit security. Then the shares of δ_a are generated from $\mathbb{K}$ and $a \cdot \delta_a$ is computed over $\mathbb{K}$. As a result, the shares of θ_a also belongs to $\mathbb{K}$. In addition, we need to modify the Input_Rand protocol. In the current implementation, one shared random bit t_i is used to perform the secure permutation. Since the shares of a and δ_a are in different fields, we also need to generate shares of t_i in $\mathbb{K}$ to permute $\langle [\theta_a], [\theta_b] \rangle$ and $\langle [\delta_a], [\delta_b] \rangle$. This change can be easily incorporated into the current implementation. Similarly, we need to modify how to generate and verify the authenticated shares for the rest of the protocol. For instance, the shares of all θ and δ values need to be generated from $\mathbb{K}$. At step 5, verifying $[[\tau]]$ would also be performed in $\mathbb{K}$. Making these changes is straightforward.

Removing The Slack

It is relatively easy to see that smaller SMC data types minimize communication and computation overhead and improve the overall performance of SMC calculations. As a result, the work in this research enables small SMC data types by eliminating the slack between the inputs and the actual size of the data types utilized in SMC computations.

For example, in previous work [96], the slack is necessary to account for the statistical parameter in the dishonest majority situation. In practice, this statistical parameter is set to at least 40 bits to assure security. As a consequence, the SMC computation uses data types that are at least 40 bits bigger than the input values. therefore, previous work necessitates 128-bit data types for SMC computations, which are required to enable 64-bit computations. Our comparison technique, on the other hand, achieves precise comparison without any slack and hence works with smaller data types. It is worth mentioning that Rabbit also eliminated the slack requirement.

Achieving Higher Statistical Security

The state of the art Rabbit capitalizes on the idea of a mixed circuit and employs edaBit to accomplish that goal. The edabit utilizes a cut-and-choose approach to assure the consistency of the produced random values. In addition, Cut-and-choose requires statistical security, which is restricted to 40-bits in recent MP-SPDZ implementation [29]. As a result, the security of Rabbit is restricted by 40-bit security in the dishonest majority setting. Our compiler, on the other hand, is not bound by this constraint. Under the HE SEAL library [30], it can handle up to 60-bit security, which can be upgraded to enable additional bit security if necessary.

4.6 Performance Evaluation

We empirically analyze the computational overhead of our proposed protocol against the state of the art comparison protocol (Rabbit) [62]. We implemented Rabbit using the state of the art MP-SPDZ library [29] and we benchmark using the TopGear protocol [13] of the SPDZ compiler. Our protocol uses Microsoft SEAL library to run the FHE of the BFV scheme [30]. Each data entry is secretly shared based on the additive secret sharing scheme for the setting of 2, 3, 4 or 5 servers. We used the Chameleon Cloud [119] to run our experiments, and each machine that represents one of the servers has the following specifications:

- Operating system: Ubuntu, Version = 18.04 LTS (Bionic Beaver).
- Machines hardware: x86_64 Cores: 12; Threads per core 2; Model: Intel®Xeon ®CPU E5-2670 v3 @ 2.30GHz, RAM 128GB.
- The ping between machines was 0.33 milliseconds.

Before presenting the results, we point out the main parameters that affect the security in both Rabbit and our proposed approach. In our solution, besides the compu-

tational security parameter λ of the underling encryption scheme, we only have the soundness security parameter κ similar to the Snd_sec parameter of SPDZ for the same purpose. SPDZ has two additional security parameters:

- ZK_sec: the statistical distance between the coefficients of the ring elements in an honest ZK protocol transcript and the ones produced by a simulation.
- DD_sec: the statistical distance between the coefficients of the ring elements in the distributed decryption protocol and the ones generated from an uniform distribution.

The three parameters are usually set to the same value, e.g., κ, to achieve κ-bit statistical security.

In addition to these parameters, Rabbit requires to generate the special random shared value which is associated with it is random shared bits based on edabit [28]. In the case of the edabit, the security parameter is $s(sec)$ represents the failure probability of the edabit protocol produces at least one incorrect edabit. It is worth mentioning that MP-SPDZ fixes this parameter to only 40 bits. This limits the security of Rabbit to only 40 bits even when setting other statistical security parameters to a higher value. On the other hand, our protocol enjoys up to 60-bit security.

4.6.1 Runtime

We examine the runtime of the secure comparison protocol based on various parameter values. The FHE computational security parameter λ was set to 128, and we set the remaining parameters as follows: $\kappa = Snd\_sec = ZK\_sec = DD\_sec = \{40, 50, 60\}$ to achieve 40, 50 or 60-bit statistical security respectively. To match the same security level, we also vary the plaintext modulus $\log_2(p) = \{40, 50, 60\}$ to match the corresponding security level related to the authenticated shares. Since edabit utilizes a cut and choose technique to ensure the consistency of random generated values,

another parameter can affect the computation overhead is the bucket size B which we set to 5 for all the tests.

Figure 4.1 (produced based on the data given in Table 4.8) shows the run-time for 40-bit security. Our approach performs around 5 times faster than the Rabbit for different number of parties. It is worth noting that for every new party joining the experiment, the run-time get almost doubled in Rabbit as well as in our protocol.

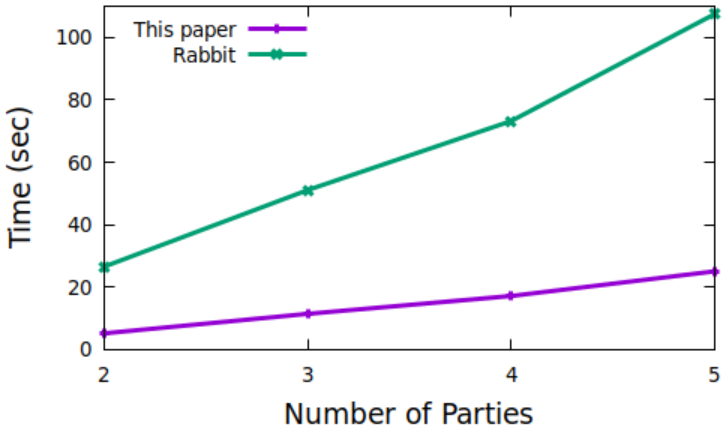

Figure 4.1: Runtime for 40-bit security

Similarly, Figures 4.2 and 4.3 (produced based on numbers in Tables 4.8) show the runtime where the parameters set to 50, 60-bit security consecutively. Despite the fact that Rabbit security is limited by the edabit security which is fixed to 40 in MP-SPDZ, changing the size of the bit security, does not affect our protocol performance and the figures suggest that we still show around 5 times faster that Rabbit in both cases of 50, and 60-bit security.

Table 4.8: Runtime for 40, 50, 60-bits security

Sec	This paper2	This paper3	This paper4	This paper5	Rabbit2	Rabbit3	Rabbit4	Rabbit5
40	5.17	11.42	17.14	25.05	26.54	51.07	73.21	107.49
50	6.13	12.71	18.88	26.94	33.81	60.71	84.46	119.34
60	8.10	14.59	21.80	30.67	64.45	105.54	146.39	189.35

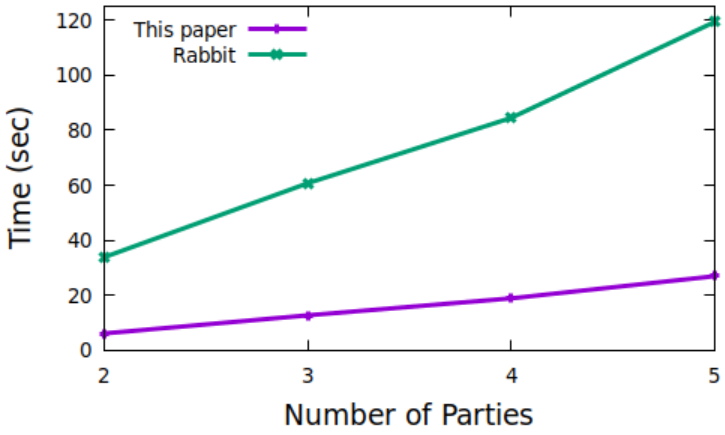

Figure 4.2: Runtime for 50-bit security

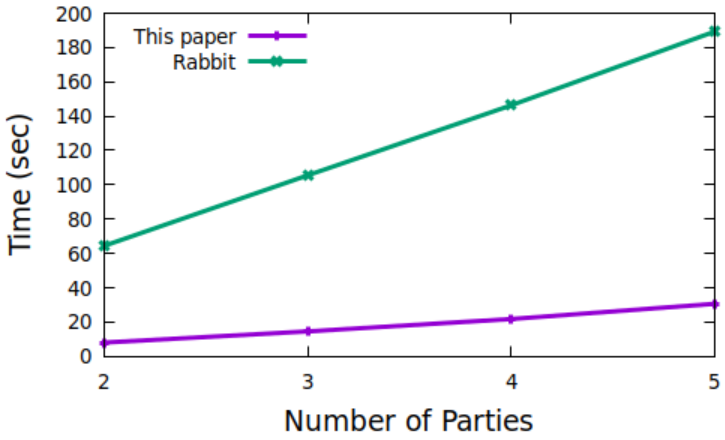

Figure 4.3: Runtime for 60-bit security

4.6.2 Communication Cost and FHE Parameters

We evaluate the amount of data communicated during protocol execution of one comparison. We vary the bit security and the number of parties similar to the previous section. Figure 4.4 (produced based on numbers in Table 4.9) depicts the data sent for 40-bit security. The figure shows that our protocol requires to send 5 to 14 times less data than Rabbit based on the number of parties.

On the other hand, figures 4.5 and 4.6 (produced based on numbers in Table 4.9) show that our protocol needs to exchange 12 up to 14 times less data than Rabbit for 50 and 60-bit security consecutively. From all the given figures, we see that every new

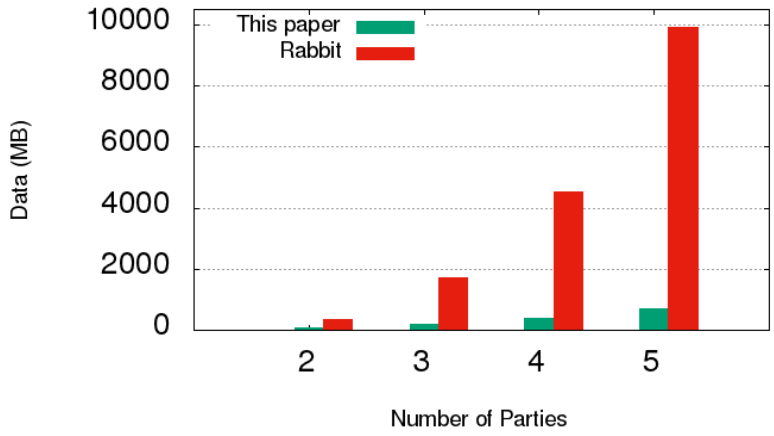

Figure 4.4: Data amount transferred among the parties for 40-bit security

added party costs Rabbit to more than double and even quadruple the amount of data to be sent in some scenarios. On the other hand, our protocol shows a consistence growth in the amount of data to be sent which get tripled for 3 parties and keeps reducing for 4 and 5 parties where the amount of data gets increased less than 2 times on average.

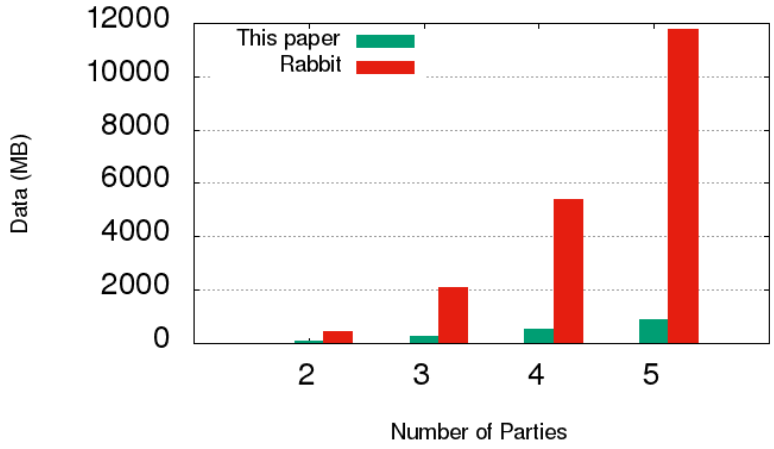

Figure 4.5: Data amount transferred among the parties for 50-bit security

To provide additional insights, Tables 4.10 and 4.11 show different security parameter settings and their effects on the underlying FHE ciphertext sizes. In all the given tables, we fix the computational security parameter $\lambda = 128$. We set the plain-

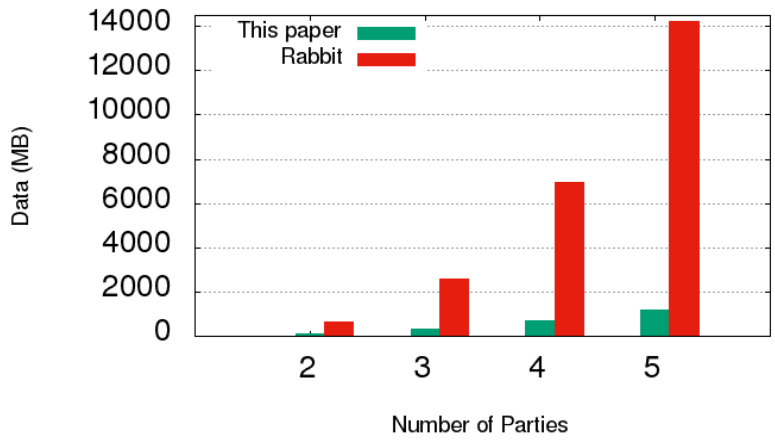

Figure 4.6: Data amount transferred among the parties for 60-bit security

text modulus to $\log_2(p) = \{40, 50, 60\}$. It is worth noting that the SPDZ utilizes two ciphertext moduli $q_1 = p_0 \cdot p_1$ and $q_0 = p_0$, where p_0 and p_1 are prime numbers. The largest modulo q_1 is used to generate level one ciphertext. Then the modulus switching is utilized to move from level one to level zero ciphertext which is associated with q_0. More information on these parameters can be found in [122, 123]. As show in the table, the ciphertext size is over 100 bit larger than ours on average which is a key factor contributing to their large communication complexity.

Table 4.9: Data amount transferred among the parties for different bit security

Sec	This paper2	This paper3	This paper4	This paper5	Rabbit2	Rabbit3	Rabbit4	Rabbit5
40	59	191	391	691	346.288	1713.28	4519.05	9912.04
50	89	253	528	904	444.955	2082.08	5396.93	11789.6
60	110	350	687	1170	621.434	2604.79	6977.77	14205.3

Table 4.10: FHE parameter sizes for this paper

$\log_2(p)$	λ	$\kappa(sec)$	N	$log_2(q)$
40	128	40	8192	218
50	128	50	8192	218
60	128	60	8192	218

Table 4.11: FHE parameter sizes for TopGear

TopGear							
$\log_2(p)$	λ	Snd_sec	ZK_sec	DD_sec	N	$\log_2(p_0)$	$\log_2(p_1)$
40	128	40	40	40	32768	136	90
50	128	50	50	50	32768	166	110
60	128	60	60	60	32768	196	130

Chapter 5

Conclusion

In this work, we present two multiparty secure comparison protocols. First SC protocol is secure against malicious minority in which the adversary controls the minority of the parties. The other SC protocol is secure under malicious majority where the adversary controls up to $n-1$ parties. Comparing to the current state of the art, the proposed protocol is much more efficient. To validate our claim, we conducted extensive empirical analyses that confirmed the efficiency of our protocol in both computation and communication. In chapter 3, we present an efficient secure comparison protocol. The protocol requires at least 3 parties with at least 2 honest parties. The protocol can be adopted to guarantee security under malicious minority model using our compiler presented in chapter 4.

In chapter 4, we present an novel approach "randomized replication" that enables n parties to compare two integers a, b in a privacy-preserving manner with collusion up to $n-1$ party. Our approach combines secure multiparty computation as well as fully homomorphic encryption to guarantee privacy against $n-1$ corrupted parties. We analyze the complexity of the proposed approach as well as its security. A comprehensive experimental evaluation shows the effectiveness of the approach in terms of both computation and communication cost.

5.1 Future Work

There are several important research directions or tasks that may further improve the quality of the work presented in this thesis as follows:

(a) Mixed protocol: as shown by the existing work, a mixed-protocol implementation (e.g., combining OT extension and homomorphic encryption) is more efficient. We will explore this direction and verify if it benefits our protocol.

(b) Parallelization: our solution is highly parallel. We plan to examine if the run-time can be further reduced by executing the protocol on a GPU.

(c) Communication optimization: in our current implementation, message sending and receiving are sequential. Since many of these messages are independent and can be sent in parallel, using more advanced communication mechanisms will likely improve the run-time.

(d) Transforming other semi-honest SC protocols: there exist many different designs for semi-honest SC. It would be very valuable to provide a systematization of these de- signs along with the proposed transformation to identify the most efficient solutions under various scenarios.

(e) Another important feature of SPDZ is its ability to split the overall computations into offline pre-processing and online computation. Although this is theoretically possible, we were not able to find such an option in Scale-Mamba [124] and MP- SPDZ [29]. Based on the existing claims, it is possible that the online phase itself would be more efficient than our protocol. In this regard, we plan to investigate strategies as SPDZ to separate the computations of the compiler into offline and online phases, and hopefully, the online phase would be much more efficient than our current solution without significantly affecting the overall performance.

Appendix A

Numerical Examples of Comparison Protocols

To help in understanding the proposed protocols we provide step by step examples. In this section, we offer two example executions of our protocol from Section 3.4 and section 3.5. In this example we will operate on values to compare $a = 10, b = 7$ and $r = 5$. Clearly in both cases our protocol should return $f = 0$ shared among the parties indicating that $a \geq b$. In the following explanation we focus on the main aspects of our approach to the comparison, and leave aside some of the details of the sharing and mapping between sharing groups which we include for security or efficiency. All values will be addressed by the steps in which they occur with the notation in accordance with the variables in the algorithm.

A.1 Example for Algorithm 8 from Section 3.4

Given $a = 10$ and $b = 7$ are secretly shared between P_1 and P_2, and it returns two shares$[c_0]$. For ease of exposition, we discarded the role of τ_i, δ and ζ as they are not going to affect the correctness of the protocol. The protocol works as follows:

1. P_3 generates a random number $r = 5$ and its shares in $\mathbb{Z}_N$, and sends the shares of r to P_1 and P_2 respectively.

2. P_1 and P_2:

   (a) Compute $c = 2(a - b) = 6$

   (b) Use the shares of r to mask c as $\eta = c + r = 11$ and send the result to the other party.

   (c) Reveal η and compute secret shares of $e = \eta \oplus r$.

   $e = 1110$

   (d) Derive secret shares of γ_i, as defined in Equation 3.5.

   $\gamma_i = 2143$

   (e) Send them γ_i to P_3.

3. P_3 reconstructs $\hat{\gamma}_i$, derives the comparison results $f = \eta_0 \oplus r_0 = 0$, and sends shares of f to P_1 and P_2.

4. P_1 and P_2 de-randomize f to derive shares of $c_0 = 0$.

A.2 Example for Algorithm 9 from Section 3.5

- *Input Transformation:* Steps 1-5 We subtract a from b and multiply the result by 2 to get the value of c. Then we add r to the later value to mask it before we open it. After opening the η, we add one. The reason for this is to capture the case where $\eta = r$ in which our comparison protocol before adding the one would be incorrect for c_0 relying on:

$$h = \begin{cases} 1 & \text{if } \eta > r \\ 0 & \text{if } \eta \leq r \end{cases}$$

Next, steps 6 and 7 focus on producing the bitwise exclusive or of $\hat{\eta}$ and r

- $e_i = 01001$

It is worth noting that we are trying to observe the most significant bit i^* where $\hat{\eta} \neq r$. We red color this bit to draw the reader attention.

- Steps 7 and 8 transform e' into $\hat{e}$, such that
  - $\hat{e}_i = 0$ for $i^* < i < l$, and
  - $\hat{e}_i \neq 0$ for $0 \leq i \leq i^*$
  - $\hat{e} = 01112$
- Step 9: The parties execute the Not_Zero protocol and the result is stored in f where

$$f_i = \begin{cases} 0 & \text{for } i^* < i < l \\ 1 & \text{for } 0 \leq i \leq i^* \end{cases}$$

  - $f_i = 01111$
- Steps 10-11: The parties locally derive the shares of $[g_i]$, such that

$$g_i = \begin{cases} 0 & \text{for } i \neq i^* \wedge 0 \leq i < l \\ 1 & \text{for } i = i^* \end{cases}$$

  - $g_i = 01000$
- Step 12: The parties locally derive the shares of $[h]$:

$$h = \begin{cases} 1 & \text{if } \eta \geq r \\ 0 & \text{if } \eta < r \end{cases}$$

As discussed early, from step 6, the computations are performed on $\hat{\eta}$. If η were used, it would result

$$h = \begin{cases} 1 & \text{if } \eta > r \\ 0 & \text{if } \eta \leq r \end{cases}$$

Consequently, this would lead to an incorrect result for c_0 at the next step.

  - $h_i = 01000$
  - $h = 1$

- Step 13: The parties derive the shares of $[c_0]$. Since $e'_0 = \hat{\eta}_0 \oplus r_0$ and $\hat{\eta} = \eta + 1$, we have $e_0 = \eta_0 \oplus r_0 = 1 - e'_0$ and c_0 can be derived accordingly:

$$c_0 = \begin{cases} e_0 = 1 - e'_0 & \text{if } h = 1 \\ 1 - e_0 = e'_0 & \text{if } h = 0 \end{cases}$$

This matches the result given in Equation 3.2.

Appendix B

Secret Sharing Numerical Examples

In this section we provide numerical examples for both replicated additive secret sharing as well as Shamir secret sharing schemes. The examples illustrate dealing two secret data then carry addition and multiplication operation. For simplicity, we set the number of parties to three which satisfies the semi-honest, honest majority and dishonest majority setting requirements.

B.1 Replicated Secret Secret Sharing Example

Generating Shares Given two secrets $a = 5, b = 8$, a dealer wishes to deal his secrets to three parties such that any three shares can open the secret. Specifically, we assume that the dealer sets $p = 61$ and progresses by selecting three random values for each secret such that :

$$5 = (10 + 12 + 44) \mod 61$$

$$8 = (22 + 35 + 12) \mod 61$$

Then the dealer sends the shares as in table B.1 such that each p_j holds a pair of shares for every secret $(a^j, a^{j-1}), (b^j, b^{j-1})$ for a and b consecutively.

Table B.1: Shares of a, b distributed to 3 participants

Party	P_1	P_2	P_3
$[a]$	(10, 44)	(12, 10)	(44, 12)
$[b]$	(22, 12)	(35, 22)	(12, 35)

Reconstructing the Secret To reconstruct a secret, the parties send the specified shares to an authorized user as described in section 2.3.3. Therefore, the user can sum up the shares to rebuild the secret back:

$$(10 + 12 + 44) \mod 61 = 5$$

$$(22 + 35 + 12) \mod 61 = 8$$

Addition Addition is done locally, therefore, there is no need for communication between the parties. Every party can locally sum the shares of $[a], [b]$ as in table B.2. To verify that the shares represent $a + b$, we can reopen the share as done in the reconstruction:

$$(32 + 47 + 56) \mod 61 = 13$$

Table B.2: Shares of $a + b$ for participants

Party	P1	P2	P3
[a+b]	(32, 56)	(47, 32)	(56, 47)

Multiplication The multiplication of two secretly shared values requires the parties to generate correlated randomness such that $r^1 + r^2 + r^3 \mod p = 0$. We assume that the parties hold the given shares with the correlated random values as shown in table B.3. Next each party locally computes the pairwise multiplication and

Table B.3: Shares of a, b and correlated randomness r for participants

Party	P1	P2	P3
$[a]$	(10, 44)	(12, 10)	(44, 12)
$[b]$	(22, 12)	(35, 22)	(12, 35)
r	33	25	3

add the correlated random value as follows:

$$((10 * 22) + (10 * 12) + (44 * 22) + 33) \mod 61 = 60$$

$$((12 * 35) + (12 * 22) + (10 * 35) + 25) \mod 61 = 22$$

$$((44 * 12) + (44 * 35) + (12 * 12) + 3) \mod 61 = 19$$

Table B.4: Shares of $a \cdot b$

Party	P1	P2	P3
$[ab]$	(60, 19)	(22, 60)	(19, 22)

Referring back to our example, we should get $5 * 8 = 40$ which can be checked by reconstructing the shares such that:

$$(60 + 22 + 19) \mod 61 = 40$$

B.2 Shamir Secret Sharing Example

Generating Shares Given two secrets $a = 5, b = 8$, a dealer wishes to deal his secrets to three parties such that any three shares can open the secret. The dealer select a random polynomial of one degree for each secret. Specifically, we assume that the dealer sets $p = 61$ and the polynomials are as follows:

$$f(x) = 5 + 45x \mod 61 \tag{B.1}$$

$$f(x) = 8 + 50x \mod 61 \tag{B.2}$$

Tables B.5 and B.6 show the dealer view of the shares of both a and b based on equations B.1, B.2.

Table B.5: Deal shares of value a

$Index$	1	2	3
$f(x) = 5 + 45x \mod 61$	50	34	18

Table B.6: Deal shares of value b

x	1	2	3
$f(x) = 8 + 50x \mod 61$	58	47	36

Next, the dealer distributes the shares such that each index refers to a specified party as shown in table B.7.

Table B.7: Shares of a, b Distributed to 3 Participants

Party	P_1	P_2	P_3
$[a]$	50	34	18
$[b]$	58	47	36

Reconstructing the Secret In order to rebuild the secret back from the shares, we generate the Lagrange coefficients as in equation B.3.

$$\beta_i = \prod_{t=1;t\neq i}^{n} \frac{-x_t}{x_i - x_t} \tag{B.3}$$

Despite the fact that based on the example setup we can reopen the secret using at least two share. However, in this example we reconstruct the secret using three shares to keep thing consistent with the number of parties given in the system. It is easy to verify that the shares hold the secret by multiplying each share with Lagrange coefficient then sum the values up such that:

$$((50 * 3) + (34 * 58) + (18 * 1)) \mod 61 = 5$$

$$((58 * 3) + (47 * 58) + (36 * 1)) \mod 61 = 8$$

Addition Addition is done without the need for communication between the parties. Every party can locally sum the shares of $[a], [b]$ as in table B.8. To verify that the shares represent $a + b$, we can reopen the share as don in the reconstruction:

$$((47 * 3) + (20 * 58) + (54 * 1)) \mod 61 = 13$$

Table B.8: Shares of $a + b$ for participants

Party	P1	P2	P3
[a+b]	47	20	54

Multiplication Multiplication under Shamir secret sharing is more involved than other secret sharing schemes as it requires an additional step to reduce the polynomial as explained in section 2.3.2. The first step, parties locally compute their share product as in table B.9.

Table B.9: Local shares multiplication for 3 participants

Party	P_1	P_2	P_3
$[ab]$	33	12	38

Next each party can locally generate the Lagrange coefficients as shown in table B.10. Then each party generates a random polynomial of degree one which is used to reshare the local product as illustrated in tale B.11. Before distributing the shares each party multiply the local shares with corespondent Lagrange coefficient β_i, for example p_1 in column 2 multiplies his share with $\beta_i = 3$, while p_2 in column 3 multiplies his share with $\beta_i = 58$, etc. for the rest of the parties. At this point parties exchange the shares and each party sums up the share as in last row of table B.11.

Table B.10: Lagrange coefficients

Party	P1	P2	P3
β_i	3	58	1

Table B.11: Deal shares of local ab

Party	P_1	P_2	P_3
$f(x) = 33 + 42x$	14	56	37
$f(x) = 12 + 34x$	46	19	53
$f(x) = 38 + 20x$	58	17	37

Table B.12: Distributed shares of $[c] = [ab]$

Party	P_1	P_2	P_3
$[w] = [u]\beta_i$	42	46	50
$[w] = [u]\beta_i$	45	4	24
$[w] = [u]\beta_j$	58	17	37
$[c] = [ab]$	23	6	50

Referring to our example, we expect to get $5 * 8 = 40$. This is can be verified via reconstructing the shares in last row of table B.11 as such:

$$((23 * 3) + (6 * 58) + (50 * 1)) \mod 61 = 40$$

B.3 Additive Secret Sharing Example

Constructing Shares Following the previous examples, assume that a dealer wants to deal his two secrets $a = 5, b = 8$ to three parties, such that any three shares can open the secret. The dealer selects two random values to be sent to first and second party. The dealer then adds up the previous random values and subtracts the sum from the original secret. Then the dealer sends the resulted value from the previous step to the third party. Assuming that for a that P_1 and P_2 shares are 19 and 31 respectively. Then the share of $p_3 = (5 - (19 + 31)) \mod 61$ which equals 16. For the other secret b, let P_1 and P_2 shares are 22 and 28 consecutively. Therefore, the share of $p_3 = (8 - (22 + 28)) \mod 61$ which gives 19. Table B.13 illustrates the distribution of the shares among three parties.

Table B.13: Shares of a, b distributed to 3 participants

Party	P_1	P_2	P_3
$[a]$	19	31	16
$[b]$	22	28	19

Reconstructing the Secret To reopen a secret back, the parties send their shares to an authorized user. Therefore, the user can adds up the shares to rebuild the secret back:

$$(19 + 31 + 16) \mod 61 = 5$$

$$(22 + 28 + 19) \mod 61 = 8$$

Addition Addition is carried locally, as a result, there is no need for communication between the parties. Each party can add the shares of $[a]$ and $[b]$ as in table B.14. To verify that the shares represent $a + b$, we can reconstruct the share as done in previous section B.3:

$$(41 + 59 + 35) \mod 61 = 13$$

Table B.14: Shares of $a + b$ for participants

Party	P1	P2	P3
[a+b]	41	59	35

Bibliography

[1] Andrew Chi-Chih Yao. Protocols for secure computations. In *Foundations of Computer Science*, volume 82, pages 160–164. IEEE, 1982.

[2] Andrew C. Yao. How to generate and exchange secrets. In *Proceedings of the 27th IEEE Symposium on Foundations of Computer Science*, pages 162–167. IEEE, 1986.

[3] Yan Huang, David Evans, Jonathan Katz, and Lior Malka. Faster secure two-party computation using garbled circuits. In *The 20th USENIX Security Symposium*, August 2011.

[4] Pascal Paillier. Public key cryptosystems based on composite degree residuosity classes. In *Advances in Cryptology - Eurocrypt '99 Proceedings, LNCS 1592*, pages 223–238, Prague, Czech Republic, May 2-6 1999. Springer-Verlag.

[5] Ivan Damgård, Martin Geisler, and Mikkel Krøigaard. Efficient and secure comparison for on-line auctions. In *Australasian conference on information security and privacy*, pages 416–430. Springer, 2007.

[6] Adi Shamir. How to share a secret. *Commun. ACM*, 22(11):612–613, November 1979.

[7] Ivan Damgård, Matthias Fitzi, Eike Kiltz, Jesper Buus Nielsen, and Tomas Toft. Unconditionally secure constant-rounds multi-party computation for equality,

comparison, bits and exponentiation. In Shai Halevi and Tal Rabin, editors, *Theory of Cryptography*, pages 285–304, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

[8] Tord Ingolf Reistad and Tomas Toft. Secret sharing comparison by transformation and rotation. In *International Conference on Information Theoretic Security*, pages 169–180. Springer, 2007.

[9] Takashi Nishide and Kazuo Ohta. Multiparty computation for interval, equality, and comparison without bit-decomposition protocol. In *International Workshop on Public Key Cryptography*, pages 343–360. Springer, 2007.

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

[11] Ivan Damgård, Marcel Keller, Enrique Larraia, Valerio Pastro, Peter Scholl, and Nigel P Smart. Practical covertly secure mpc for dishonest majority–or: breaking the spdz limits. In *European Symposium on Research in Computer Security*, pages 1–18. Springer, 2013.

[12] Marcel Keller, Valerio Pastro, and Dragos Rotaru. Overdrive: Making spdz great again. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 158–189. Springer, 2018.

[13] Carsten Baum, Daniele Cozzo, and Nigel P Smart. Using topgear in overdrive: A more efficient zkpok for spdz. In *International Conference on Selected Areas in Cryptography*, pages 274–302. Springer, 2019.

[14] Oded Goldreich. *The Foundations of Cryptography*, volume 2, chapter General Cryptographic Protocols. Cambridge University Press, 2004.

[15] Yonatan Aumann and Yehuda Lindell. Security against covert adversaries: Efficient protocols for realistic adversaries. In Salil P. Vadhan, editor, *Theory of Cryptography*, pages 137–156, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg.

[16] Yonatan Aumann and Yehuda Lindell. Security against covert adversaries: Efficient protocols for realistic adversaries. *Journal of Cryptology*, April 7 2009.

[17] Rafail Ostrovsky and Moti Yung. How to withstand mobile virus attacks. In *Proceedings of the tenth annual ACM symposium on Principles of distributed computing*, pages 51–59, 1991.

[18] Ran Canetti and Amir Herzberg. Maintaining security in the presence of transient faults. In *Annual International Cryptology Conference*, pages 425–438. Springer, 1994.

[19] 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.

[20] Mihaela Ion, Ben Kreuter, Erhan Nergiz, Sarvar Patel, Shobhit Saxena, Karn Seth, David Shanahan, and Moti Yung. Private intersection-sum protocol with applications to attributing aggregate ad conversions. Cryptology ePrint Archive, Report 2017/738, 2017. `https://eprint.iacr.org/2017/738`.

[21] andCurv(www.curv.co). UnboundTech.(www.unboundtech.com), Sepior(sepior.com).

[22] https://sharemind.cyber.ee. Sharemind.

[23] https://duality.cloud Duality.

[24] Dan Bogdanov, Marko Jõemets, Sander Siim, and Meril Vaht. How the estonian tax and customs board evaluated a tax fraud detection system based on secure multi-party computation. In *International conference on financial cryptography and data security*, pages 227–234. Springer, 2015.

[25] Dan Bogdanov, Liina Kamm, Baldur Kubo, Reimo Rebane, Ville Sokk, and Riivo Talviste. Students and taxes: a privacy-preserving study using secure computation. *Proceedings on Privacy Enhancing Technologies*, 2016(3):117–135, 2016.

[26] 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.

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

[28] Daniel Escudero, Satrajit Ghosh, Marcel Keller, Rahul Rachuri, and Peter Scholl. Improved primitives for mpc over mixed arithmetic-binary circuits. In *Annual International Cryptology Conference*, pages 823–852. Springer, 2020.

[29] Marcel Keller. Mp-spdz: A versatile framework for multi-party computation. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, CCS '20, page 1575âĂŞ1590, New York, NY, USA, 2020. Association for Computing Machinery.

[30] Microsoft SEAL (release 3.6), November 2020. Microsoft Research, Redmond, WA.

[31] Rosario Gennaro, Michael O. Rabin, and Tal Rabin. Simplified VSS and fast-track multiparty computations with applications to threshold cryptography. In *Proceedings of the* 17^{th} *Annual ACM Symposium on Principles of Distributed Computing*, pages 101–111, September 21 1998.

[32] Zuzana Beerliová-Trubíniová and Martin Hirt. Efficient multi-party computation with dispute control. In Shai Halevi and Tal Rabin, editors, *Theory of Cryptography*, pages 305–328, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

[33] Martin Hirt and Jesper Buus Nielsen. Robust Multiparty Computation with Linear Communication Complexity. In *Advances in Cryptology - CRYPTO 2006*, volume 4117, pages 463–482. Springer Berlin Heidelberg, Berlin, Heidelberg, 2006. Series Title: Lecture Notes in Computer Science.

[34] Ivan Damgård and Jesper Buus Nielsen. Scalable and Unconditionally Secure Multiparty Computation. In Alfred Menezes, editor, *Advances in Cryptology - CRYPTO 2007*, pages 572–590. Springer Berlin Heidelberg, Berlin, Heidelberg, 2007. Series Title: Lecture Notes in Computer Science.

[35] Zuzana Beerliová-Trubíniová and Martin Hirt. Perfectly-secure mpc with linear communication complexity. In Ran Canetti, editor, *Theory of Cryptography*, pages 213–230, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.

[36] Vipul Goyal, Yanyi Liu, and Yifan Song. Communication-Efficient Unconditional MPC with Guaranteed Output Delivery. In Alexandra Boldyreva and Daniele Micciancio, editors, *Advances in Cryptology âĂŞ CRYPTO 2019*, pages 85–114, Cham, 2019. Springer International Publishing. Series Title: Lecture Notes in Computer Science.

[37] Vipul Goyal, Yifan Song, and Chenzhi Zhu. Guaranteed-Output Delivery Comes Free in Honest Majority MPC. In *Advances in Cryptology âĂŞ CRYPTO 2020*. Springer International Publishing, 2020. Series Title: Lecture Notes in Computer Science.

[38] Eli Ben-Sasson, Serge Fehr, and Rafail Ostrovsky. Near-Linear Unconditionally-Secure Multiparty Computation with a Dishonest Minority. In Reihaneh Safavi-Naini and Ran Canetti, editors, *Advances in Cryptology âĂŞ CRYPTO 2012*, Lecture Notes in Computer Science, pages 663–680, Berlin, Heidelberg, 2012. Springer.

[39] Yehuda Lindell and Ariel Nof. A Framework for Constructing Fast MPC over Arithmetic Circuits with Malicious Adversaries and an Honest-Majority. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 259–276, Dallas Texas USA, October 2017. ACM.

[40] Donald Beaver. Efficient multiparty protocols using circuit randomization. In Joan Feigenbaum, editor, *Advances in Cryptology — CRYPTO '91*, pages 420–432, Berlin, Heidelberg, 1992. Springer Berlin Heidelberg.

[41] Koji Chida, Daniel Genkin, Koki Hamada, Dai Ikarashi, Ryo Kikuchi, Yehuda Lindell, and Ariel Nof. Fast Large-Scale Honest-Majority MPC for Malicious Adversaries. In Hovav Shacham and Alexandra Boldyreva, editors, *Advances in Cryptology âĂŞ CRYPTO 2018*, pages 34–64. Springer International Publishing, Cham, 2018. Series Title: Lecture Notes in Computer Science.

[42] Daniel Genkin, Yuval Ishai, Manoj M. Prabhakaran, Amit Sahai, and Eran Tromer. Circuits resilient to additive attacks with applications to secure computation. In *Proceedings of the Forty-Sixth Annual ACM Symposium on Theory*

of Computing, STOC '14, pages 495–504, New York, NY, USA, 2014. Association for Computing Machinery.

[43] Daniel Genkin, Yuval Ishai, and Antigoni Polychroniadou. Efficient multi-party computation: From passive to active security via secure simd circuits. In Rosario Gennaro and Matthew Robshaw, editors, *Advances in Cryptology – CRYPTO 2015*, pages 721–741, Berlin, Heidelberg, 2015. Springer Berlin Heidelberg.

[44] Donald Beaver. Efficient multiparty protocols using circuit randomization. In *Annual International Cryptology Conference*, pages 420–432. Springer, 1991.

[45] 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.

[46] Zvika Brakerski and Vinod Vaikuntanathan. Fully homomorphic encryption from ring-lwe and security for key dependent messages. In *Annual cryptology conference*, pages 505–524. Springer, 2011.

[47] Jesper Buus Nielsen, Peter Sebastian Nordholt, Claudio Orlandi, and Sai Sheshank Burra. A new approach to practical active-secure two-party computation. In *Annual Cryptology Conference*, pages 681–700. Springer, 2012.

[48] Enrique Larraia, Emmanuela Orsini, and Nigel P Smart. Dishonest majority multi-party computation for binary circuits. In *Annual Cryptology Conference*, pages 495–512. Springer, 2014.

[49] Ivan Damgård and Sarah Zakarias. Constant-overhead secure computation of boolean circuits using preprocessing. In *Theory of Cryptography Conference*, pages 621–641. Springer, 2013.

[50] Ivan Damgård, Rasmus Lauritsen, and Tomas Toft. An empirical study and some improvements of the minimac protocol for secure computation. In *International Conference on Security and Cryptography for Networks*, pages 398–415. Springer, 2014.

[51] Tore Kasper Frederiksen, Marcel Keller, Emmanuela Orsini, and Peter Scholl. A unified approach to mpc with preprocessing using ot. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 711–735. Springer, 2015.

[52] Tore Kasper Frederiksen, Marcel Keller, Emmanuela Orsini, and Peter Scholl. A unified approach to mpc with preprocessing using ot.

[53] Marcel Keller, Emmanuela Orsini, and Peter Scholl. Mascot: Faster malicious arithmetic secure computation with oblivious transfer. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, CCS '16, page 830âĂŞ842, New York, NY, USA, 2016. Association for Computing Machinery.

[54] Niv Gilboa. Two party rsa key generation. In *Annual International Cryptology Conference*, pages 116–129. Springer, 1999.

[55] Ivan Damgård, Matthias Fitzi, Eike Kiltz, Jesper Buus Nielsen, and Tomas Toft. Unconditionally secure constant-rounds multi-party computation for equality, comparison, bits and exponentiation. In *Theory of Cryptography Conference*, pages 285–304. Springer, 2006.

[56] Securescm analysis. public deliverable d9.2, securescm project. July 2009.

[57] Ian F Blake and Vladimir Kolesnikov. Strong conditional oblivious transfer and computing on intervals. In *International Conference on the Theory and*

Application of Cryptology and Information Security, pages 515–529. Springer, 2004.

[58] Tord Ingolf Reistad and Tomas Toft. Secret sharing comparison by transformation and rotation. In *International Conference on Information Theoretic Security*, pages 169–180. Springer, 2007.

[59] Tord Ingolf Reistad. Multiparty comparison-an improved multiparty protocol for comparison of secret-shared values. In *International Conference on Security and Cryptography*, volume 1, pages 325–330. SCITEPRESS, 2009.

[60] Tord Reistad and Tomas Toft. Linear, constant-rounds bit-decomposition. In *International Conference on Information Security and Cryptology*, pages 245–257. Springer, 2009.

[61] Berry Schoenmakers and Pim Tuyls. Efficient binary conversion for paillier encrypted values. In Serge Vaudenay, editor, *Advances in Cryptology - EUROCRYPT 2006*, pages 522–537, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

[62] Eleftheria Makri, Dragos Rotaru, Frederik Vercauteren, and Sameer Wagh. Rabbit: Efficient comparison for secure multi-party computation. *IACR Cryptol. ePrint Arch.*, 2021:119, 2021.

[63] Craig Gentry. *A fully homomorphic encryption scheme*. Stanford university, 2009.

[64] Craig Gentry. Fully homomorphic encryption using ideal lattices. In *Proceedings of the 41st annual ACM symposium on Symposium on theory of computing - STOC '09*, page 169, Bethesda, MD, USA, 2009. ACM Press.

[65] Craig Gentry and Shai Halevi. Implementing gentryâĂŹs fully-homomorphic encryption scheme. In *Annual international conference on the theory and applications of cryptographic techniques*, pages 129–148. Springer, 2011.

[66] Nigel P Smart and Frederik Vercauteren. Fully homomorphic encryption with relatively small key and ciphertext sizes. In *International Workshop on Public Key Cryptography*, pages 420–443. Springer, 2010.

[67] Marten van Dijk, Craig Gentry, Shai Halevi, and Vinod Vaikuntanathan. Fully homomorphic encryption over the integers. Cryptology ePrint Archive, Report 2009/616, 2009. `https://eprint.iacr.org/2009/616`.

[68] Jean-Sébastien Coron, Avradip Mandal, David Naccache, and Mehdi Tibouchi. Fully homomorphic encryption over the integers with shorter public keys. In *Annual Cryptology Conference*, pages 487–504. Springer, 2011.

[69] Jean-Sébastien Coron, David Naccache, and Mehdi Tibouchi. Public key compression and modulus switching for fully homomorphic encryption over the integers. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 446–464. Springer, 2012.

[70] Jung Hee Cheon, Jean-Sébastien Coron, Jinsu Kim, Moon Sung Lee, Tancrede Lepoint, Mehdi Tibouchi, and Aaram Yun. Batch fully homomorphic encryption over the integers. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 315–335. Springer, 2013.

[71] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (Leveled) fully homomorphic encryption without bootstrapping. In *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference (ITCS'12)*, pages 309–325, January 2012.

[72] Zvika Brakerski and Vinod Vaikuntanathan. Efficient Fully Homomorphic Encryption from (Standard) LWE. In *2011 IEEE 52nd Annual Symposium on Foundations of Computer Science*, pages 97–106, Palm Springs, CA, USA, October 2011. IEEE.

[73] Zvika Brakerski. Fully Homomorphic Encryption without Modulus Switching from Classical GapSVP. volume 7417, pages 868–886. Berlin, Heidelberg, 2012. Series Title: Lecture Notes in Computer Science.

[74] Junfeng Fan and Frederik Vercauteren. Somewhat Practical Fully Homomorphic Encryption. 2012:19, 2012.

[75] Craig Gentry, Amit Sahai, and Brent Waters. Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based. In *Annual Cryptology Conference*, pages 75–92. Springer, 2013.

[76] Shai Halevi and Victor Shoup. Bootstrapping for helib. In *Annual International conference on the theory and applications of cryptographic techniques*, pages 641–670. Springer, 2015.

[77] Léo Ducas and Daniele Micciancio. Fhew: bootstrapping homomorphic encryption in less than a second. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 617–640. Springer, 2015.

[78] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachene. Faster fully homomorphic encryption: Bootstrapping in less than 0.1 seconds. In *international conference on the theory and application of cryptology and information security*, pages 3–33. Springer, 2016.

[79] Jung Hee Cheon, Andrey Kim, Miran Kim, and Yongsoo Song. Homomorphic encryption for arithmetic of approximate numbers. In *International Conference*

on the Theory and Application of Cryptology and Information Security, pages 409–437. Springer, 2017.

[80] Jung Hee Cheon, Kyoohyung Han, Andrey Kim, Miran Kim, and Yongsoo Song. A full rns variant of approximate homomorphic encryption. Cryptology ePrint Archive, Report 2018/931, 2018. `https://eprint.iacr.org/2018/931`.

[81] Hao Chen, Ilaria Chillotti, and Yongsoo Song. Improved bootstrapping for approximate homomorphic encryption. Cryptology ePrint Archive, Report 2018/1043, 2018. `https://eprint.iacr.org/2018/1043`.

[82] Gilad Asharov, Yehuda Lindell, Thomas Schneider, and Michael Zohner. More efficient oblivious transfer and extensions for faster secure computation. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, pages 535–548, 2013.

[83] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game, or a completeness theorem for protocols with honest majority. In *Providing Sound Foundations for Cryptography: On the Work of Shafi Goldwasser and Silvio Micali*, pages 307–328. 2019.

[84] Sai Sheshank Burra, Enrique Larraia, Jesper Buus Nielsen, Peter Sebastian Nordholt, Claudio Orlandi, Emmanuela Orsini, Peter Scholl, and Nigel P Smart. High performance multi-party computation for binary circuits based on oblivious transfer.

[85] Yehuda Lindell and Ben Riva. Blazing fast 2pc in the offline/online setting with security for malicious adversaries. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, pages 579–590, 2015.

[86] Benny Pinkas, Thomas Schneider, Gil Segev, and Michael Zohner. Phasing: Private set intersection using permutation-based hashing. In *24th {USENIX} Security Symposium ({USENIX} Security 15)*, pages 515–530, 2015.

[87] Benny Pinkas, Thomas Schneider, and Michael Zohner. Faster private set intersection based on {OT} extension. In *23rd {USENIX} Security Symposium ({USENIX} Security 14)*, pages 797–812, 2014.

[88] Yuval Ishai, Manoj Prabhakaran, and Amit Sahai. Secure arithmetic computation with no honest majority. In *Theory of Cryptography Conference*, pages 294–314. Springer, 2009.

[89] Carsten Baum, Ivan Damgård, Tomas Toft, and Rasmus Zakarias. Better preprocessing for secure multiparty computation. In *International Conference on Applied Cryptography and Network Security*, pages 327–345. Springer, 2016.

[90] Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, Lisa Kohl, and Peter Scholl. Efficient pseudorandom correlation generators: Silent ot extension and more. In *Annual International Cryptology Conference*, pages 489–518. Springer, 2019.

[91] Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, Lisa Kohl, Peter Rindal, and Peter Scholl. Efficient two-round ot extension and silent non-interactive secure computation. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, pages 291–308, 2019.

[92] Yuval Ishai, Joe Kilian, Kobbi Nissim, and Erez Petrank. Extending oblivious transfers efficiently. In *Annual International Cryptology Conference*, pages 145–161. Springer, 2003.

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

[94] Payman Mohassel and Peter Rindal. Aby3: A mixed protocol framework for machine learning. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pages 35–52, 2018.

[95] Toshinori Araki, Assi Barak, Jun Furukawa, Marcel Keller, Yehuda Lindell, Kazuma Ohara, and Hikaru Tsuchida. Generalizing the spdz compiler for other protocols. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pages 880–895, 2018.

[96] Dragos Rotaru and Tim Wood. Marbled circuits: Mixing arithmetic and boolean circuits with active security. In *International Conference on Cryptology in India*, pages 227–249. Springer, 2019.

[97] Niklas Büscher, Daniel Demmler, Stefan Katzenbeisser, David Kretzmer, and Thomas Schneider. Hycc: Compilation of hybrid protocols for practical secure computation. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pages 847–861, 2018.

[98] Muhammad Ishaq, Ana L Milanova, and Vassilis Zikas. Efficient mpc via program analysis: A framework for efficient optimal mixing. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, pages 1539–1556, 2019.

[99] Nishanth Chandran, Divya Gupta, Aseem Rastogi, Rahul Sharma, and Shardul Tripathi. Ezpc: programmable and efficient secure two-party computation for machine learning. In *2019 IEEE European Symposium on Security and Privacy (EuroS&P)*, pages 496–511. IEEE, 2019.

[100] Abdelrahaman Aly, Emmanuela Orsini, Dragos Rotaru, Nigel P Smart, and Tim Wood. Zaphod: Efficiently combining lsss and garbled circuits in scale.

In *Proceedings of the 7th ACM Workshop on Encrypted Computing & Applied Homomorphic Cryptography*, pages 33–44, 2019.

[101] Dragos Rotaru, Nigel P Smart, Titouan Tanguy, Frederik Vercauteren, and Tim Wood. Actively secure setup for spdz. *IACR Cryptol. ePrint Arch.*, 2019:1300, 2019.

[102] Charlotte Bonte, Nigel P Smart, and Titouan Tanguy. Thresholdizing hashed-dsa: Mpc to the rescue. *International Journal of Information Security*, pages 1–16, 2021.

[103] Elette Boyle, Niv Gilboa, and Yuval Ishai. Function secret sharing. In *Annual international conference on the theory and applications of cryptographic techniques*, pages 337–367. Springer, 2015.

[104] Elette Boyle, Niv Gilboa, and Yuval Ishai. Secure computation with preprocessing via function secret sharing. In *Theory of Cryptography Conference*, pages 341–371. Springer, 2019.

[105] Ronald L Rivest, Adi Shamir, and Leonard Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, 1978.

[106] 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.

[107] Peter W Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM review*, 41(2):303–332, 1999.

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

[109] Oriol Farras and Carles Padró. Ideal hierarchical secret sharing schemes. *IEEE transactions on information theory*, 58(5):3273–3286, 2012.

[110] Ivan Damgård and Rune Thorbek. Linear integer secret sharing and distributed exponentiation. In *International Workshop on Public Key Cryptography*, pages 75–90. Springer, 2006.

[111] Dan Bogdanov. Foundations and properties of shamirâĂŹs secret sharing scheme research seminar in cryptography. *University of Tartu, Institute of Computer Science May 1st*, 2007.

[112] Rosario Gennaro, Michael O Rabin, and Tal Rabin. Simplified vss and fast-track multiparty computations with applications to threshold cryptography. In *Proceedings of the seventeenth annual ACM symposium on Principles of distributed computing*, pages 101–111. ACM, 1998.

[113] Thijs Veugen, Frank Blom, Sebastiaan JA de Hoogh, and Zekeriya Erkin. Secure comparison protocols in the semi-honest model. *IEEE Journal of Selected Topics in Signal Processing*, 9(7):1217–1228, 2015.

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

[115] Junfeng Fan and Frederik Vercauteren. Somewhat practical fully homomorphic encryption. *IACR Cryptol. ePrint Arch.*, 2012:144, 2012.

[116] Abdelrahaman Aly, K Cong, D Cozzo, M Keller, E Orsini, D Rotaru, O Scherer, P Scholl, NP Smart, T Tanguy, et al. Scale–mamba v1. 14: Documentation, 2021.

[117] Tord Reistad. MULTIPARTY COMPARISON - An Improved Multiparty Protocol for Comparison of Secret-shared Values. In *Proceedings of the International Conference on Security and Cryptography*, pages 325–330, Milan, Italy, 2009. SciTePress - Science and and Technology Publications.

[118] Sameer Wagh, Divya Gupta, and Nishanth Chandran. Securenn: 3-party secure computation for neural network training. *Proc. Priv. Enhancing Technol.*, 2019(3):26–49, 2019.

[119] Chameleon cloud. `https://www.chameleoncloud.org/`.

[120] Yuval Ishai, Manoj Prabhakaran, and Amit Sahai. Founding cryptography on oblivious transfer – efficiently. In *Advances in Cryptology – CRYPTO 2008*, pages 572–591, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.

[121] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *Proceedings of the 42nd IEEE Symposium on Foundations of Computer Science*, pages 136–145. IEEE, 2001.

[122] Scale-mamba software. `https://homes.esat.kuleuven.be/~nsmart/SCALE/`.

[123] Martin Albrecht, Melissa Chase, Hao Chen, Jintai Ding, Shafi Goldwasser, Sergey Gorbunov, Shai Halevi, Jeffrey Hoffstein, Kim Laine, Kristin Lauter, Satya Lokam, Daniele Micciancio, Dustin Moody, Travis Morrison, Amit Sahai, and Vinod Vaikuntanathan. Homomorphic encryption security standard. Technical report, HomomorphicEncryption.org, Toronto, Canada, November 2018.

[124] A. Aly, M. Keller, E. Orsini, D. Rotaru, P. Scholl, N.P. Smart, and T. Wood. Scale-mamba v1.12: Documentation (2021), 2021.

VITA

Ali Ataeemh Al-lami was born and raised in Amarah city, Mysan, Iraq. He obtained the Bachelor degree in Computer Science from Shatt Alarab Uinveristy, Basrah, Iraq. Ali gained 3 years of experience in the industry at Zain and Asiacell telecommunication companies. As well as, 3 years in the education field at Mysan University. His passion for research made him enroll into the MS program for Computer Science at informatics institute of technology , Baghdad, Iraq, where he gained his master degree in computer science. After one year, he won a scholarship of the Higher Committee of Education Development (HCED) which funded his PhD program in the United States for five years. Through that, he got started his PhD program at Missouri University of Science and Technology, Rolla, Missouri,USA under the guidance of professor Dr. Wei Jiang. Later, he transferred to the University of Missouri, Columbia, USA along with his advisor. He is currently completing a PhD at the University of Missouri and is working in researching privacy-preserving data analytics with Dr. Wei Jiang.