Maliciously Secure Multiparty Computation Protocols and Its Applications

March 2023

Hikaru Tsuchida

Maliciously Secure Multiparty Computation Protocols and Its Applications

Graduate School of Science and Technology
Degree Programs in Systems and Information Engineering
University of Tsukuba

March 2023

Hikaru Tsuchida

Abstract

Secure computation aims to compute a target function while keeping parties' input hidden and outputting only the computation result. Secure computation has been attracting attention because it can be used to securely realize outsourced computation and cross-organizational data collaboration. Cryptographic techniques for secure computation is divided into two schemes: *multiparty computation* (MPC) and *homomorphic encryption*. We focus on MPC schemes.

MPC is a cryptographic technology that enables a set of parties to compute an arbitrary function represented as a circuit without revealing any information other than the output. MPC schemes enable parties to compute the target function securely even if an adversary corrupts some of the parties. Some of the well-known MPC-based applications and services are machine learning algorithms (e.g., the evaluation of deep neural networks and decision trees) and biometric authentication (e.g., iris authentication). MPC-based applications and services are considered useful because they can securely process confidential information.

There are two models for the rate at which the adversary can corrupt the parties: *honest majority* and *dishonest majority*. We let n and t be the number of parties and corrupted parties, respectively. In the honest majority setting, it holds that $t < n/2$. In the dishonest majority setting, it holds that $t < n$. There are also two types of the adversary's behavior: *semi-honest* and *malicious*. Parties corrupted by a semi-honest adversary attempt to learn as much information as they can while following the protocol specifications. Parties corrupted by a malicious adversary can attempt to not only learn as much information as they can but also manipulate the computation results while they are allowed to behave arbitrarily. They may also attempt to launch a denial-of-service (DoS) attack. In particular, there are two types of typical security requirements related to the delivery of outputs against malicious adversaries: *fairness* and *robustness*. Fairness ensures that the parties corrupted by the malicious adversary can receive their outputs if and only if the honest parties also receive their outputs. Robustness ensures that all honest parties always receive their outputs regardless of the malicious adversaries' behavior without aborting the protocol. Note that the MPC protocols with fairness may be aborted when detecting cheats by corrupted parties. Hence, the MPC protocols with fairness cannot prevent DoS attacks by the malicious adversary while the MPC protocols with robustness can. In other words, robustness is the stronger security requirement than fairness. Achievable security requirement varies depending on the MPC protocols. Most MPC protocols have a trade-off between the security achieved and the performance.

Secret-sharing-based MPC (SS-MPC) is one of the most common types of MPC. In SS-MPC protocols, each party distributes its inputs to multiple parties as secret *shares* that look like random values over an algebraic structure (e.g., a residue ring). Parties use shares locally and communicate among parties without revealing their inputs to compute functions. Most SS-MPC protocols have a trade-off between the communication and round complexities.

In recent years, SS-MPC schemes over the ring among the constant and small number of parties have been actively studied. In particular, it with $t < n/3$ can achieve both practical

performance and strong security requirements like fairness or robustness.

However, most of ring-based SS-MPC schemes with $t < n/3$ achieving fairness or robustness focus to improve the communication or round complexities of the share multiplication and subprotocols for the evaluation of deep neural networks, not to improve it of the subprotocols for the evaluation of decision trees and iris authentication. To the best of our knowledge, the SS-MPC protocols for the evaluation of decision trees and iris authentication over the ring achieving both practical performance and fairness or robustness have not been proposed.

In this paper, we propose the ring-based SS-MPC protocol for the evaluation of decision trees with fairness and constant rounds and that for the iris authentication with robustness. Our study contributes to constructing MPC-based applications and services securely and efficiently in the real world.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Secure computation aims to compute a target function while keeping parties' input hidden and outputting only the computation result. Secure computation has been attracting attention because it can be used to securely realize outsourced computation and cross-organizational data collaboration. Cryptographic techniques for secure computation is divided into two schemes: *multiparty computation* (MPC) and *homomorphic encryption*. We focus on MPC schemes.

MPC is a cryptographic technology that enables a set of parties to compute an arbitrary function represented as a circuit without revealing any information other than the output [2–4]. MPC schemes enable parties to compute the target function securely even if an adversary corrupts some of the parties. Some of the well-known MPC-based applications and services are machine learning algorithms (e.g., the evaluation of deep neural networks and decision trees) and biometric authentication (e.g., iris authentication). MPC-based applications and services are considered useful because they can securely process confidential information.

Many MPC protocols have trade-offs with respect to security from the following perspectives:

- **Number of corrupted parties by an adversary.** There are two types of models: *honest majority* and *dishonest majority*. We denote the number of parties and corrupted parties as n and t, respectively. In the honest majority setting, $t < n/2$ holds. In the setting of dishonest majority, $t < n$ holds.

- **Adversary's computing power.** The adversary's computing power is roughly divided into two types: computationally *bounded* and *unbounded*. We call the security against a computationally bounded and unbounded adversary *computational security* and *information-theoretic security*, respectively.

- **Adversary's behavior.** Adversaries can be divided into two types from the perspective of possible behaviors of corrupted parties: *semi-honest* (a.k.a passive) adversary and *malicious* (a.k.a active) adversary. Parties corrupted by a semi-honest adversary attempt to learn as much information about private inputs of other parties as they can while following the protocol specifications. Parties corrupted by a malicious ad-

versary can also attempt to learn as much information as they can. In addition, they can manipulate the computation results while they are allowed to behave arbitrarily. They may also attempt to launch a denial-of-service (DoS) attack.

In particular, there are two types of typical security requirements related to the delivery of outputs against malicious adversaries: *fairness* and *robustness* (a.k.a *guaranteed output delivery*, GOD). Fairness ensures that the parties corrupted by the malicious adversary can receive their outputs if and only if the honest parties also receive their outputs. Robustness ensures that all honest parties always receive their outputs regardless of the malicious adversaries' behavior without aborting the protocol. Note that the MPC protocols with fairness may be aborted when detecting cheats by corrupted parties[1]. Hence, the MPC protocols with fairness cannot prevent DoS attacks by the malicious adversary while the MPC protocols with robustness can. In other words, robustness is the stronger security requirement than fairness. Achievable security requirement varies depending on the MPC protocols. Most MPC protocols have a trade-off between the security achieved and the performance.

Secret-sharing-based MPC (SS-MPC) [2, 3] is one of the most common types of MPC. In SS-MPC protocols, each party distributes its inputs to multiple parties as secret *shares* that look like random values over an algebraic structure (e.g., a residue ring). Parties use shares locally and communicate among parties without revealing their inputs to compute functions. Most SS-MPC protocols have a trade-off between the communication and round complexities.

In recent years, SS-MPC schemes over the ring among the constant and small number of parties have been actively studied [5–18]. In particular, it with $t < n/3$ can achieve both practical performance and strong security requirements like fairness or robustness.

However, most of ring-based SS-MPC schemes with $t < n/3$ achieving fairness or robustness focus to improve the communication or round complexities of the share multiplication and subprotocols for the evaluation of deep neural networks, not to improve it of the subprotocols for the evaluation of decision trees and iris authentication. To the best of our knowledge, the SS-MPC protocols for the evaluation of decision trees and iris authentication over the ring achieving both practical performance and fairness or robustness have not been proposed.

In this paper, we propose the ring-based SS-MPC protocol for the evaluation of decision trees with fairness and constant rounds [1, 19] and that for the iris authentication with robustness [20, 21]. Our study contributes to constructing MPC-based applications and services securely and efficiently in the real world.

This paper is organized as follows. §2 prepares preliminaries for the other chapters. §3 shows the overview of typical MPC aaplications and the need to research PDTE and biometric authentication based on SS-MPC protocols. §4 describes in detail the contributions with respect to the evaluation of decision trees with fairness and constant rounds [1, 19]. To describe it, §4 also describes in detail the contributions with respect to the evaluation

[1] *Security with abort* is a weaker security notion than fairness. If an MPC protocol achieving security with abort detects cheating, it aborts. That is, security with abort does *not* ensure that the parties corrupted by the malicious adversary can receive their outputs if and only if the honest parties also receive their outputs.

of decision trees with semi-honest security and constant rounds [22, 23]. §5 explains the contributions for the iris authentication with robustness [20, 21]. Finally, we conclude and explain the future work in §6.

Chapter 2

Preliminaries

2.1 Notations

We let $\mathbb{Z}_2$, $\mathbb{Z}_{2^k}$, $\mathbb{Z}_p(=\mathbb{F}_p$, where p is prime) and $\mathbb{Z}_{p'}(=\mathbb{F}_{p'}$, where p' is the smallest prime larger than k) be the residue rings modulo 2, 2^k, p or p'. We denote the share vectors and inner-products as $\vec{a} = (a_0, \ldots, a_{k-1}), \vec{b} = (b_0, \ldots, b_{k-1})$, and $\vec{a} \cdot \vec{b} = \sum_{j=0}^{k-1} a_j \cdot b_j \mod L$, respectively, where $a_j, b_j \in \mathbb{Z}_L$ $(j = 0, \ldots, k-1)$ and $L = 2,\ 2^k, p$ or p'.

We let $\oplus$ and $\cdot$ be the XOR operator and AND operator, respectively. Note that we also use $\cdot$ as the multiplication operator on $\mathbb{Z}_L$ where $L = 2,\ 2^k, p$ or p'.

Let P_i be the i-th party. P_i has a collision-resistant hash function. We let $H_{i'}$ $(i' = 0,\ 1,\ 2)$ be the clients who are the helper entities. We denote i mod 3 by $\overline{i}$. For example, $P_{\overline{i+1}}$ means $P_{(i+1) \bmod 3}$. We also denote i mod 4 by $\underline{i}$. For example, $P_{\underline{i+1}}$ means $P_{(i+1) \bmod 4}$.

The security parameter is denoted by κ. The κ-bit bit string is $\{0,1\}^\kappa$. We use the (cryptographically secure) pseudo-random functions (PRF) $F_L : \{0,1\}^\kappa \times \{0,1\}^\kappa \to \mathbb{Z}_L$ where $L = 2,\ 2^k, p$ or p'. $H_{i'}$ has $\mathsf{seed} \in \{0,1\}^\kappa$. We use the unique identifier, $\mathsf{uid}, \mathsf{vid}_j, \mathsf{vid}_j^{(\sigma)}, \mathsf{vid}_j^{(\sigma_1)}, \mathsf{vid}_j^{(\mu_1)}, \mathsf{vid}_j^{(1)}, \ldots, \mathsf{vid}_j^{(N-1)} \in \{0,1\}^\kappa$. We use the public unique identifier, e.g., counter values. That is, any parties and clients can know these values. We also use $F^*_{p'} : \{0,1\}^\kappa \times \{0,1\}^\kappa \to \mathbb{Z}^*_{p'} (= \mathbb{Z}_{p'} \setminus \{0\})$.

Let $v|_j \in \mathbb{Z}_2$ be j-th bit of $v \in \mathbb{Z}_{2^k}$. We also denote by $v|_{j,\ldots,i} \in \mathbb{Z}_{2^k}$ the part of bit strings of $v \in \mathbb{Z}_{2^k}$ from $j(\geq i)$-th bit to i-th bit. Let $\mathsf{msb}(v)$ be the MSB of v. For example, if $v = 1100_{(2)} = 12 \in \mathbb{Z}_{2^4}$, we have that $v|_0 = 0$, $v|_1 = 0$, $v|_2 = 1$, $v|_3 = \mathsf{msb}(a) = 1$, and $v|_{2,\ldots,0} = 100_{(2)} = 4$.

We denote the set of permutations of an array that has M elements on $\mathbb{Z}_L$ by S_M.

2.2 Collision-resistant Hash Function

For checking the message consistency efficiently, we use the collision-resistant hash function $\mathcal{H}$. Note that the cryptographic hash function is required to have the following three properties:

1. **Preimage resistance**: It is (computationally) hard to compute the value x from

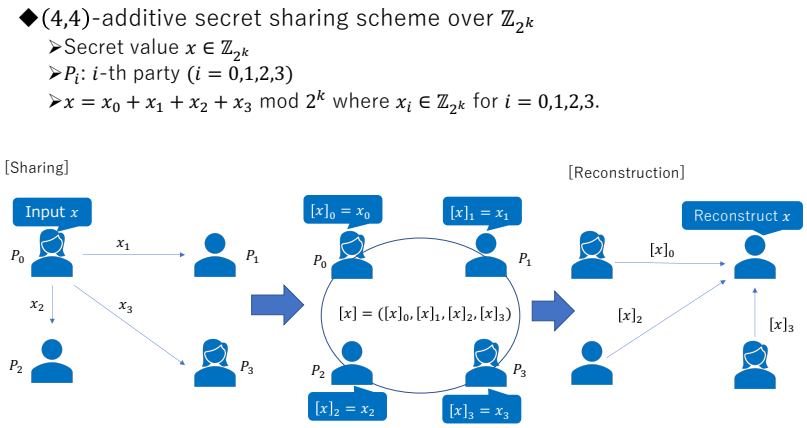

Figure 2.1: Toy example of secret sharing scheme

$\mathcal{H}(x)$.

2. **Collision resistance**: It is (computationally) hard to find the pair (x, y) such that $\mathcal{H}(x) = \mathcal{H}(y)$ and $x \neq y$.

3. **Second preimage resistance**: Given the value y, it is (computationally) hard to compute the value x such that $\mathcal{H}(x) = \mathcal{H}(y)$ and $x \neq y$. Note that $\mathcal{H}$ has the second preimage resistance if $\mathcal{H}$ has the collision resistance.

$\mathcal{H}$ that we used has the collision resistance.

2.3 Overview of Secret Sharing

Secret sharing is a cryptographic technology that distributes secret values among multiple parties and manages it securely. Secret values are distributed over an algebraic structure, and distributed values are indistinguishable from random numbers. We call the distributed values *shares*.

Unless a certain number of shares are gathered, the original secret value cannot be reconstructed from shares. Let n and t be the number of parties and the threshold value, respectively. For example, a secret sharing such that the secret value is reconstructed when $t+1$ shares out of n shares are gathered is called $(t+1, n)$-threshold secret sharing scheme. As a toy example, we show (n, n)-additive secret sharing scheme, a type of $(t+1, n)$-threshold secret sharing scheme in Fig. 2.1. Fig. 2.1 shows the $(4, 4)$-additive secret sharing scheme over $\mathbb{Z}_{2^k}$. Let $[x]$ be the shares of x among parties. We also let $[x]_i$ be the i-th party's share.

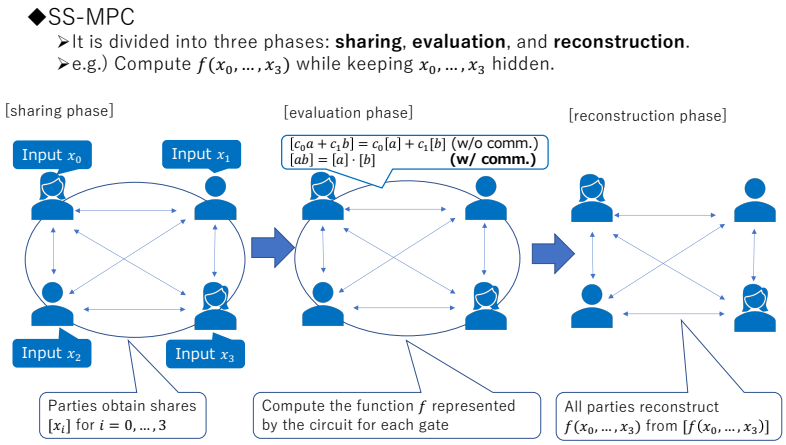

Figure 2.2: Overview of SS-MPC

In Fig. 2.1, P_0 and P_3 are the input dealer and the reconstructing party, respectively. Note that $[x]_i (= x_i)$ is indistinguishable from a random value over $\mathbb{Z}_{2^k}$. Hence, no one can reconstruct the secret value x, or even obtain partial information about x, unless the four $[x]_i$ are gathered.

In this paper, we use a variety of secret sharing schemes. Since the secret sharing schemes used in each chapter are different, see the preliminaries of each chapter for details.

2.4 Overview of SS-MPC

SS-MPC is one of the typical MPC schemes. It uses a secret sharing scheme to keep the parties' inputs confidential. Fig. 2.2 shows the overview of SS-MPC.

SS-MPC is divided into three phases: *sharing*, *evaluation*, and *reconstruction* phases. In sharing phase, parties distribute their inputs as shares among parties. Then, parties compute the function represented by the circuit gate-by-gate without reconstructing shares in evaluation phase. In most SS-MPC schemes, the scalar addition/multiplication and the share addition do not need communications between parties. The share multiplication needs communications between parties. Finally, parties obtain the computation results by reconstructing it from shares in reconstruction phase.

Chapter 3

Overview of Typical MPC Applications

In this chapter, we provide some examples of typical MPC applications.

1. **Cryptographic key management.** A hardware security module (HSM) is a device that securely manages cryptographic keys. Users can perform encryption, decryption, and signing operations while never taking their keys out of HSMs. A virtual HSM (vHSM) is a software realization of the HSM by MPC schemes. A vHSM is superior to an HSM in terms of flexibility. For example, Unbound Security, which Coinbase acquired, sold the vHSM product [24]. As another example, Sepior, which Blockdaemon acquired, sold the cryptographic key management system for distributed management of cryptographic keys. That is, MPC eliminates single points of failure by managing cryptographic keys in a distributed manner.

2. **Secure database and privacy-preserving data analysis.** MPC can enable database processing while keeping each data and query hidden. As described in [24], for example, Galois, Inc. developed the Jana system that provides a private data as a service (PDaaS) system for relational data by integrating MPC, differential privacy, and searchable encryption. As another example, Cybernetica developed Sharemind, a privacy-preserving database based on an SS-MPC scheme [25–27].

   The MPC enables not only simple database operations but also data analysis securely. In the Estonian Association of Information Technology and Telecommunications (ITL), member companies' economic benchmarking data were collected periodically and analyzed using MPC to promote their business. The MPC ensures that data on each company's economic indicators are kept confidential so that each company does not know the data of the others. Another example is the privacy-preserving statistical studies on linked database (PRIST), MPC enabled us to link and analyze education and tax data while reducing privacy risks.

3. **Privacy-preserving auction.** In Danish sugar beet auctions [28], the buyer specifies the quantity they wish to purchase for some possible price. Sellers also select the

amounts they want to sell for a given potential price. MPC realizes the auction so that supply and demand match while keeping their bids confidential. It allows sugar beet production contracts to be securely realized between farmers and sugar companies. As another example, Energiauktion.dk provides power procurement as a secure SaaS-type auction by MPC.

4. **Training/inference of neural network models.** In recent years, many works on secure training/inference of the neural network model using MPC have been proposed [10,12–14,16–18,29–36]. In the secure training/inference of the neural network model, MPC conceals model parameters of neural networks, training data, and test data. It allows sensitive data to be used for model training and inference. It also protects model parameters, which are intellectual property, from being leaked.

5. **Training/inference of decision tree models.** As in the training/inference of the neural network models, in that of the decision tree models, MPC conceals model parameters of trees, training data, and test data. There have been studies on training/inference of decision tree models [37–40] proposed in recent years, although fewer than those on that of neural network models.

6. **Biometric matching(, identification, or authentication).** MPC-based biometric template matching can achieve matching while keeping extracted and registered features hidden. It is relatively easy to implement because MPC performs simple distance calculations, and prior research exists [41–43].

Additionally, several studies [44,45] have been done on MPC and law regarding establishing the above applications in the real world. In [45], Helminger and Rechberger discuss the role of MPC in the EU General Data Protection Regulation (GDPR) [46]. Treiber et al. discuss the role of the MPC in the exchange of private information between legal enforcement agencies from both a technical and legal perspective [44].

In this paper, we focus on outsourced computations or cooperative computations by multiple participants. Hence, this paper does not cover cryptographic key management.

For the secure database and privacy-preserving data analysis, the priority is still on performance when processing large amounts of data and resolving issues related to the flexibility of queries and analysis. Since the priorities associated with MPC security requirements, such as fairness and robustness, are low. Therefore, we leave the improving the trade-off between security and performance for these applications as future works.

For the training/inference of neural network models, there have been many works in recent years using MPC schemes with a few constant parties, which improves the trade-off between performance and the security requirements achieved. However, despite its usefulness for tabular data, such as those owned by companies, rather than neural networks, there is little research on the trade-off between performance and the security requirements (such as fairness or robustness) achieved for the training/inference of decision tree models.

There are many works on the performance of biometric matching. However, DoS attacks should be a concern when it is used for authentication. Therefore, it should achieve not only high performance but also robustness.

Hence, we focus on the inference of decision tree models with fairness and biometric matching with robustness. These applications are practical even if the amount of data handled or the size of a single piece of data is small. Applications such as secure databases, privacy-preserving data analysis and auction, training and inference of neural network models, and training of decision trees are difficult to achieve practical use as MPC applications because of the huge amount of data handled and the large size of each piece of data. Therefore, we leave improving the efficiency and safety of these applications as a future issue.

Chapter 4

Private Decision Tree Evaluation with Constant Rounds via fair SS-4PC[2]

4.1 Introduction

4.1.1 Background

There are two types of typical MPC: *garbled circuit* (*GC*) [4, 47] and *SS-MPC* [2, 3]. Most GC protocols compute an arbitrary function represented as a binary circuit among two parties by using encrypted truth tables (garbled tables) and oblivious transfer (OT) [48, 49], which is a public-key primitive. A GC protocol requires many communication bits and small and constant communication rounds.

In SS-MPC, each party distributes its inputs, and the computation proceeds with secret shares that look like random numbers among several parties. In the SS-MPC protocol, each party computes a function, which is represented as a binary, arithmetic, or mixed circuit (composed of binary and arithmetic circuits) by using shares locally and communicating among parties. The SS-MPC protocol requires small communication bits and many communication rounds. There are two types of SS-MPC: *SS-MPC over the field* and *SS-MPC over the ring*. The two schemes differ in the mathematical structure they use.

The former [2, 50] uses a finite field. SS-MPC over the field can compute an arbitrary function represented as an arithmetic circuit. It can construct constant-round protocols by using the multiplicative inverse. However, it requires modulo operations with a large prime number. Therefore, the computational cost of SS-MPC over the field is often heavier than SS-MPC over the ring.

The latter uses a residue ring (e.g., power-of-two ring). In particular, the secure three-

[2]This chapter is based on "Private Decision Tree Evaluation with Constant Rounds via (Only) SS-3PC" [23] and "Constant-Round Fair SS-4PC for Private Decision Tree Evaluation" [1], by the same author, which appeared in the IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences, Copyright(C)2022 IEICE. The content of this chapter corresponds to references [1,19,22,23] relevant to the requirements for completion.

party computation protocol (3PC) over the ring [5] and four-party computation protocol over the ring [12, 14] have gained attention in recent years because they can perform high throughput even when they compute a complex function represented as mixed circuits. SS-MPC on the power-of-two ring benefits from not only a small communication complexity but also a small computation cost because there is no need to perform a modulo operation explicitly if it uses a power-of-two ring that equals the size of the data type. However, SS-MPC over the ring cannot easily construct the constant-round protocol due to a lack of the multiplicative inverse.

In addition, *homomorphic encryption* (*HE*) [51–56] computes the function represented as an arithmetic circuit without revealing input values, which is different from MPC. Although HE does not require communications during computation, it requires a large computation cost.

As mentioned above, GC, SS-MPC, and HE have a trade-off relationship for the computation cost, communication cost, and type of computable circuit. Hence, the optimal choice depends on the computing resources, communication environment, and function related to applications. To mitigate this problem, a hybrid scheme of GC and SS-MPC [10,12,30] and that of GC and HE [41] were proposed and studied. In particular, to mitigate the communication cost problem, the offline-online paradigm is also widely known. It divides the MPC protocol into an offline phase (where the protocol processes part of computation that is independent) and an online phase (where the protocol processes the rest of computation with parties' inputs). The offline-online paradigm can reduce the communication cost of the online phase even if it increases the communication cost of the offline phase and the whole computation. Therefore, the offline-online paradigm is useful for MPC applications that focus on the response time of queries.

One typical application of secure computation (e.g., GC, SS-MPC, and HE) that has gained attention in recent years is the private decision tree evaluation (PDTE). A decision tree is a commonly-used tool of decision support and widely studied in machine learning. The PDTE protocol outputs the (encrypted) class label assigned to the leaf node as the correct classification result without revealing confidential (or sensitive) information about the tree (e.g., decision threshold values assigned to the internal node, comparison operations assigned to the internal node, or class label assigned to the leaf node) and the input feature vector. For example, there are several applications of the PDTE: electrocardiogram classification [57, 58], and remote diagnosis [57]. Araki et al. [6] experimented with the private evaluation of a decision tree for credit decisions using 3PC over the ring.

Kiss et al. [39] published a systematization of knowledge paper about the PDTE. In [39], the PDTE is divided into three phases: *feature selection*, *comparison*, and *path evaluation*. In the feature selection phase, a feature is selected from the input feature vector without revealing the values of the input feature vector, value of the selected feature, or index. In the comparison phase, the selected feature is compared with the threshold value without revealing the selected feature, threshold value, or comparison result. In the path evaluation phase, the classification result is outputted without revealing the comparison results. In [39], Kiss et al. focused on a constant-round protocol by using GC, HE, and a hybrid scheme of GC and SS-MPC, but did not mention the constant-round protocol using only the SS-MPC.

SS-MPC over the field can provide constant-round equality-testing, less-than, and k-ary

OR[3] (AND) protocols [50]. Thus, we can easily obtain a constant-round (but somewhat less efficient) protocol for the PDTE. However, it is non-trivial to construct the PDTE protocol with constant rounds using only SS-MPC over the ring. For example, Cock et al. proposed a PDTE protocol using only the SS-MPC over the ring in [59]. However, the round complexity of their protocol is proportional to the height of the tree and the bit length of the ring.

If we use GC, HE, or SS-MPC over the field, it is trivial to construct a constant-round PDTE protocol. However, these techniques are less efficient regarding the communication complexity and computation cost than SS-MPC over the ring. On the other hand, it is non-trivial to construct the PDTE protocol with constant rounds using only SS-MPC over the ring. Therefore, the naive PDTE protocol with SS-MPC over the ring often has a larger round complexity.

There are trade-offs about communication/computational cost in GC, HE, and SS-MPC, making it difficult to choose the optimal technology, but there is currently no protocol based only on SS-MPC with constant rounds yet. Taking into account the potential risks related to technology compromise, it is also desirable to have a variety of methods to construct the constant-round PDTE protocol. For this reason, it will be meaningful to devise a construction of constant-round PDTE protocol based not only on GC, HE, and the hybrid protocols but also on (only) SS-MPC.

Thus, we ask the following: Could we construct the constant-round PDTE protocol with constant rounds using only SS-MPC over the ring?

4.1.2 Our Approach

We propose a PDTE protocol with constant rounds using the semi-honest 3PC with single corruption[4]. We propose an efficient constant-round protocol over the ring in each phase as follows.

1. We propose a more efficient most significant bit (MSB) extraction protocol with constant rounds (Protocol 4) than SecureNN [33]. Our scheme is used to construct the constant-round less-than protocol and the constant-round equality-testing protocol. Hence, our scheme can compute the process in the comparison phase with constant rounds efficiently.

2. We propose a more efficient oblivious array read (OAR) protocol over the ring (Protocol 6) than existing ones [60]. Our scheme can compute the process in the feature selection phase with constant rounds efficiently.

3. We propose a path evaluation protocol with constant rounds (Protocol 11). To the best of our knowledge, this is the first constant-round path evaluation protocol over the ring.

[3]This computes $\bigvee_{i=1}^{k} x_i$ where $x_i \in \{0, 1\}$.

[4]Similarly to existing 3PC [5, 33], our protocols can also achieve only privacy (not correctness) against a malicious adversary under the client-server model.

4. We propose a PDTE protocol with constant rounds over the ring (Protocol 13) using the above contributions. To the best of our knowledge, this is the first PDTE protocol proposed with constant rounds over the ring without GC, HE, or OT (i.e., without public-key primitives). Our scheme is based on a 3PC over the ring that has efficient communication complexity and computation cost. Therefore, our scheme can be performed efficiently even if the communication environment has a large latency and limited communication bandwidth.

We also propose a more efficient PDTE protocol with constant rounds by using semi-honest 3PC with single corruption over the field than the naive construction. We propose efficient constant-round protocols over the field in each phase as follows.

1. We propose a more efficient OAR protocol over the field (Protocol 7) than the existing ones [60]. Our scheme can compute the process in the feature selection phase with constant rounds efficiently. For more details of (theoretical) performance comparison between Protocol 7 and the existing scheme, see Table 4.2.

2. We propose a path evaluation protocol over the field with constant rounds (Protocol 12). The number of rounds of our scheme is larger than one of the naive path evaluation protocols. However, our scheme requires only the constant rounds and the smaller communication complexity than the naive path evaluation protocol. Therefore, our scheme can compute the process in the path evaluation phase with constant rounds efficiently. For more details of (theoretical) performance comparison between Protocol 12 and the existing scheme, see Table 4.4.

3. We propose two PDTE protocols with constant rounds over the field (Protocols 14 and 15) using the above contributions. Protocol 14 needs the smallest communication rounds in the online phase among the existing schemes. Protocol 15 removes the term of $2^h \cdot \log_2(h)$ from the communication complexity in the path evaluation phase where h is the height of the tree while achieving the constant rounds. For more details of (theoretical) performance comparison between Protocols 14 and 15 and the existing schemes, see Tables 4.3 and 4.4.

Table 4.1 shows the theoretical performance comparison of the MSB extraction protocol between SecureNN [33] and our scheme. Let L be an even number. The MSB extraction protocol in SecureNN [33] takes the shares over the odd ring $\mathbb{Z}_{L-1}$ as inputs and outputs the shares over the even ring $\mathbb{Z}_L$. However, SS-MPC over the ring often uses the power-of-two ring (i.e., the even ring) for computational efficiency. Hence, we should compare the performance of our scheme with the combination of a share conversion protocol that converts the shares on $\mathbb{Z}_L$ to the shares on $\mathbb{Z}_{L-1}$ and the MSB extraction protocol. The protocol combining the share conversion and MSB extraction protocols in SecureNN [33] requires 9 rounds through the overall computation. On the other hand, our scheme (that takes the shares on $\mathbb{Z}_{2^k}$ as inputs and outputs the shares on $\mathbb{Z}_2$) requires 7 rounds in the online phase and 8 rounds in the online + offline phase. If we use the bit conversion protocol [10] and convert the output of the shares of our scheme to the shares on $\mathbb{Z}_{2^k}$, the number of rounds

in the online phase is fewer than those of SecureNN. Therefore, our scheme is superior to SecureNN for round complexity.

Table 4.2 shows the theoretical performance comparison of the OAR protocol between our schemes and the existing schemes [60]. The OAR protocol takes the shared array elements and shared index value as inputs and outputs the shared array element corresponding to the index value. Existing schemes [60] take the shares of the index value on $\mathbb{Z}_m$ (where m is the length of the array) or shares of Shamir's secret sharing as input. However, we note that the share of the index value may not always be on $\mathbb{Z}_m$ as it depends on the preceding and subsequent processes of the computation. In the context of the PDTE, both shares of the index value and array elements may be over the same residue ring if the secure computing methods (GC, HE, or SS-MPC) are used for not only evaluation but also constructing the decision tree. Hence, the OAR protocol is required to be a constant-round protocol even if both shares of the index value and array elements are over the same residue ring. However, it is not clear whether the existing schemes [60] are constant-round protocols or not even if both shares of the index value and array elements are on the same residue ring because most share conversion methods over the ring require many rounds. On the other hand, our scheme over the ring (Protocol 6) is a constant-round protocol even if both shares of the index value and array elements are over the same residue ring $\mathbb{Z}_{2^k}$. Furthermore, our scheme over the field (Protocol 7) requires only 8 rounds in the online phase, which is fewer than the number of rounds of a custom three-party construction over the field [60] in the online phase. The number of rounds of our scheme over the field (Protocol 7) needs more rounds than the general construction with constant rounds [60]. However, the communication complexity of our scheme over the field (Protocol 7), $O(m \log_2(p))$, is more efficient than one of a general construction with constant rounds [60], $O(m \log_2(\log_2(m)) \log_2(p))$ even when $t = 1$ and $n = 3$. Therefore, our schemes have still an advantage compared with the existing protocols [60].

Tables 4.3, 4.4 and 4.5 show the theoretical performance comparison of the PDTE protocol between our schemes and naive constructions. The naive construction over the ring (Protocol 1 in 4.2.12) and [59] are not constant-round protocols, but our scheme over the ring (Protocol 13) is. Furthermore, the number of rounds of our scheme over the field (Protocol 14) in the online phase is fewer than those of the naive construction over the field (Protocol 2 in 4.2.12) in the online phase. In addition, the communication complexity of our another scheme over the field (Protocol 15) is smaller than the communication complexity of Protcol 14 while achieving the constant rounds. The communication complexity of Protcol 15 in the path evaluation phase is $O(2^h \cdot \log_2(p))$ because Protcol 15 uses our efficient path evaluation protocol with constant rounds (Protocol 12) in the path evaluation phase. Hence, it is smaller than the communication complexity of Protcol 14 and Protocol 2 not using Protocol 12 in the path evaluation phase, $O(2^h \cdot \log_2(h) \cdot \log_2(p))$. However, the number of communication rounds of Protcol 15 is more than the number of communication rounds of Protcol 14 and Protocol 2. That is, our schemes over the field have a trade-off relationship for the communication cost.

Furthermore, we propose a maliciously secure PDTE protocol with constant rounds

Table 4.1: Comparison of communication complexity of secure three-party MSB extraction protocol between existing protocol and ours (k: bit length of ring, $L(=2^k)$: modulus of ring, p': smallest prime number greater than k, $(2,2)$-ASS: 2-out-of-2 additive secret sharing scheme, $(2,3)$-RSS: 2-out-of-3 replicated secret sharing scheme)

	Input Share	Output Share	Rounds		Comm. [bits/all parties]	
			Offline	Online	Offline	Online
Naive MSB Extraction (using bit-decomposition protocol of ABY3 [10])	(2,3)-RSS on $\mathbb{Z}_{2^k}$	(2,3)-RSS on $\mathbb{Z}_2$	0	$1+\log_2(k)$	0	$3k$
SecureNN [33] (MSB Extraction)	(2,2)-ASS on $\mathbb{Z}_{L-1}$	(2,2)-ASS on $\mathbb{Z}_L$	0	5	0	$4k\log_2(p')+13k$
SecureNN [33] (Share Convert + MSB Extraction)	(2,2)-ASS on $\mathbb{Z}_L$	(2,2)-ASS on $\mathbb{Z}_L$	0	9	0	$8k\log_2(p')+19k$
This Work (Protocol 4)	(2,3)-RSS on $\mathbb{Z}_{2^k}$	(2,3)-RSS on $\mathbb{Z}_2$	1	7	$6k(k-1)$	$11k+3(k-1)\log_2(p')+4$
This Work (Protocol 4) + Bit Conversion of ABY3 [10]	(2,3)-RSS on $\mathbb{Z}_{2^k}$	(2,3)-RSS on $\mathbb{Z}_{2^k}$	1	8	$6k(k-1)$	$17k+3(k-1)\log_2(p')+4$

Table 4.2: Comparison of communication complexity of oblivious array read protocol between existing protocols and ours (m: length of array, k: bit length of ring, p: prime number, p': smallest prime number greater than k, n: number of parties, $t(<n/2)$: number of corrupted parties, $(t+1,n)$-ASS: $(t+1)$-out-of-n additive secret sharing scheme, $(t+1,n)$-SSS: $(t+1)$-out-of-n Shamir's secret sharing scheme, $(t+1,n)$-RSS: $(t+1)$-out-of-n replicated secret sharing scheme)

	Share of Input and Output Array Element	Share of Input Index	Rounds		Comm. [bits/all parties]	
			Offline	Online	Offline	Online
[61]	$(t+1,n)$-SSS on $\mathbb{F}_p$	$(t+1,n)$-SSS on $\mathbb{F}_p$	m	3	$mn(n+1)\log_2(p)$	$3n(n+1)\log_2(p)$
Custom Three-party Construction over Ring [60]	(2,2)-ASS on $\mathbb{Z}_{2^k}$	(2,2)-ASS on $\mathbb{Z}_m$	0	2	0	$4mk$
Custom Three-party Construction over Field [60]	(2,2)-ASS on $\mathbb{F}_p$	(2,3)-SSS on $\mathbb{F}_p$	1	10	$(6\log_2(m)-2)\cdot\log_2(p)$	$(4m+2\log_2(m)+12)\cdot\log_2(p)$
General Construction (with Constant Rounds) [60]	$(t+1,n)$-SSS on $\mathbb{F}_p$	$(t+1,n)$-SSS on $\mathbb{F}_p$	1	4	$(3m\log_2(\log_2(m))+\log_2(m)$ $-1)n(n-1)\log_2(p)$	$(m\log_2(\log_2(m))+3)n(n-1)\log_2(p)$
This Work over Ring (Protocol 6)	(2,3)-RSS on $\mathbb{Z}_{2^k}$	(2,3)-RSS on $\mathbb{Z}_{2^k}$	1	12	$6k^2\log_2(m)$	$\log_2(m)(9k+(3k-3)\log_2(p')+4)$ $+6m\log_2(m)+4mk+2m+8k$
This Work over Field (Protocol 7)	(2,3)-SSS on $\mathbb{F}_p$	(2,3)-SSS on $\mathbb{F}_p$	3	8	$18m\log_2 p$	$45m\log_2(p)$

building on Trident [12], a fair 4PC[5] [12]. Our approaches are as follows:

1. We propose maliciously secure shuffle and most significant bit (MSB) extraction protocols with fairness and constant rounds. Tables 4.6 and 4.7 show that only our protocols achieve malicious security and fairness with constant rounds. Table 4.6 also shows that the number of the communication rounds of our shuffle in the online phase is the same as semi-honest secure shuffle [62] and the number of communication bits of our shuffle in the online phase is lower than [62].

2. By using our proposed shuffle and MSB extraction along with Trident, we construct each maliciously secure protocol related to each phase with constant rounds and fairness. Table 4.8 shows that only our protocol achieves malicious security stronger than [22] with constant rounds. It also shows that the number of rounds of our protocol in the online phase is smaller than naive construction in the practical setting where the parameters m (the number of features), h (the height of tree), and k (the bit

[5]Trident [12] includes not only the SS-MPC but also the GC. We use only the SS-MPC in our protocol.

Table 4.3: Comparison of total communication complexity between naive PDTE protocols and ours (m: number of features, k: bit length of ring, p: prime number, p': smallest prime number greater than k, h: height of tree)

	Rounds		Comm. [bits/all parties]	
	Offline	Online	Offline	Online
Naive Construction over Ring (Protocol 1 in 4.2.12)	0	$\log_2(\log_2(m))$ $+2\log_2(k)+\log_2(h)+9$	0	$(2^h-1)\cdot(3\log_2 m+3\log_2(\log_2 m)+6m\log_2 m+4mk+2m$ $+17k+6k\log_2 k+3)+2^h\cdot(3h+6k-3)+3k$
Naive Construction over Field (Protocol 2 in 4.2.12)	1	18	$(2^h-1)\cdot(6\log_2(m)-2+24k$ $-18\log_2(k)-30)\cdot\log_2(p)+2^h\cdot\log_2(h)\cdot6\log_2(p)$	$(2^h-1)\cdot(4m+2\log_2(m)+6k+6\log_2(k)+36)\cdot\log_2(p)$ $+(2^h\cdot3\log_2(h)+1)\cdot6\log_2(p)$
[59]	0	$h+\log_2(k)+5$	0	$(2^h-1)\cdot(10m\log_2(m)+30k-10\log_2(k)-20)$ $+2^h\cdot10hk$
This Work over Ring (Protocol 13)	1	26	$(2^h-1)\cdot(6k^2\log_2(m)+18k^2)$	$(2^h-1)\cdot(\log_2(m)(9k+(3k-3)\log_2(p')+4)+(9k-9)\log_2(p')$ $+6m\log_2(m)+4mk+2m+35k+21)+2^h\cdot(6k+9h)$
This Work over Field (Protcol 14)	3	16	$(2^h-1)\cdot(18m+24k-18\log_2(k)$ $-30)\cdot\log_2(p)+2^h\cdot\log_2(h)\cdot6\log_2(p)$	$(2^h-1)\cdot(27m+6k+6\log_2(k)+24)\cdot\log_2(p)$ $+(2^h\cdot3\log_2(h)+1)\cdot6\log_2(p)$
This Work over Field (Protcol 15)	3	20	$(2^h-1)\cdot(18m+24k-18\log_2(k)$ $-30)\cdot\log_2(p)+2^h\cdot18\log_2(p)$	$(2^h-1)\cdot(27m+6k+6\log_2(k)+24)\cdot\log_2(p)$ $+2^h\cdot27\log_2(p)$

Table 4.4: Comparison of communication complexity between naive PDTE protocols and ours (Comm. in Offline; communication bits per all parties in the offline phase, Comm. in Online; communication bits per all parties in the online phase, m: number of features, k: bit length of ring, p: prime number, p': smallest prime number greater than k, h: height of tree)

	Feature Selection		Comparison		Path Evaluation	
	Rounds in Offline	Comm. in Offline	Rounds in Offline	Comm. in Offline	Rounds in Offline	Comm. in Offline
	Rounds in Online	Comm. in Online	Rounds in Online	Comm. in Online	Rounds in Online	Comm. in Online
Naive Construction over Ring	0	0	0	0	0	0
(Protocol 1 in 4.2.12)	$\log_2(\log_2(m))+5$	$(2^h-1)\cdot(3\log_2(m)+3\log_2(\log_2(m))$ $+6m\log_2(m)+4mk+2m+8k)$	$2\log_2(k)+2$	$(2^h-1)\cdot(6k\log_2(k)$ $+9k+3)$	$\log_2(h)+2$	$2^h\cdot(3(h-1)$ $+6k)+3k$
Naive Construction over Field	1	$(2^h-1)\cdot(6\log_2(m)$ $-2)\cdot\log_2(p)$	1	$(2^h-1)\cdot(4k-3\log_2(k)$ $-5)\cdot6\log_2(p)$	1	$2^h\cdot\log_2(h)\cdot6\log_2(p)$
(Protocol 2 in 4.2.12)	10	$(2^h-1)\cdot(4m+2\log_2(m)$ $+12)\cdot\log_2(p)$	4	$(2^h-1)\cdot(k+\log_2(k)$ $+4)\cdot6\log_2(p)$	4	$(2^h\cdot3\log_2(h)+1)\cdot6\log_2(p)$
[59]	0	0	0	0	0	0
	2	$(2^h-1)\cdot10m\log_2(m)$	$\log_2(k)+2$	$(2^h-1)\cdot(30k-10\log_2(k)-220$	$h+1$	$2^h\cdot10hk$
This Work over Ring	1	$(2^h-1)\cdot(6k^2\log_2(m))$	1	$(2^h-1)\cdot18k^2$	0	0
(Protocol 13)	12	$(2^h-1)\cdot(\log_2(m)(9k+(3k-3)\log_2(p')+4)$ $+6m\log_2(m)+4mk+2m+8k)$	10	$(2^h-1)\cdot(27k$ $+(9k-9)\log_2(p')+21)$	4	$2^h\cdot(6k+9h)$
This Work over Field	3	$(2^h-1)\cdot18m\log_2(p)$	1	$(2^h-1)\cdot(4k$ $-3\log_2(k)-5)\cdot6\log_2(p)$	1	$2^h\cdot\log_2(h)\cdot6\log_2(p)$
(Protocol 14)	8	$(2^h-1)\cdot27m\log_2(p)$	4	$(2^h-1)\cdot(k+\log_2(k)$ $+4)\cdot6\log_2(p)$	4	$(2^h\cdot3\log_2(h)+1)\cdot6\log_2(p)$
This Work over Field	3	$(2^h-1)\cdot18m\log_2(p)$	1	$(2^h-1)\cdot(4k-3\log_2(k)$ $-5)\cdot6\log_2(p)$	3	$2^h\cdot18\log_2(p)$
(Protocol 15)	8	$(2^h-1)\cdot27m\log_2(p)$	4	$(2^h-1)\cdot(k+\log_2(k)$ $+4)\cdot6\log_2(p)$	8	$2^h\cdot27\log_2(p)$

length of the modulus) are greater than $m=256$, $h=16$, and $k=2048$ respectively[6]. Therefore, our scheme can not only solve the theoretical open problem of [22] but also be run efficiently and securely even when the communication environment has a large latency.

We put the tables of theoretical communication complexities for the building blocks and the proposed protocols in §4.2.9 and §4.7.4 to help the reader understand.

[6]For example, the typical datasets for the privacy-preserving machine learning, Texas (that contains hospital discharge data [63]) and Purchase (that contains purchasing histories [64]), have 600 and 6170 features, respectively. Araki et al. [6] conducted an experiment about the private evaluation of a concrete decision tree at heights from 4 to 30 for credit decisions [65] using 3PC. The modulus size of residue ring needs to be 128, 256, 512, 1024, 2048 or even larger to guarantee the accuracy of the fixed-point calculations. If users need to train the decision tree securely via ID3 [66] and MPC before PDTE, training process contains the logarithmic calculations via MPC with guaranteed high accuracy. As another case, in the hybrid model of the decision tree and regression model (i.e., each regression model is assigned to each leaf), the larger ring size is required to guarantee high accuracy.

Table 4.5: Comparison of the number of the pseudo-random function (PRF) invocations between naive PDTE protocols and ours (# PRF. in Offline; the number of PRF invocations per all parties in the offline phase, # PRF in Online; the number of PRF invocations per all parties in the online phase, m: number of features, k: bit length of ring, h: height of tree, |#BuildingBlock *in off.*|: the number of PRF invocations for the BuildingBlock in the offline phase, |#BuildingBlock *in on.*|: the number of PRF invocations for the BuildingBlock in the online phase. We note that BuildingBlock means each building block.)

	Feature Selection	Comparison	Path Evaluation
	# PRF in Offline	# PRF in Offline	# PRF in Offline
	# PRF in Online	# PRF in Online	# PRF in Online
Naive Construction over Ring	0	0	0
(Protocol 1 in 4.2.12)	$(2^h-1)\cdot(\lvert\#\mathsf{NaiveBitDec}\rvert + \log_2(m)\lvert\#\mathsf{BitConversion}\rvert + \lvert\#\mathsf{rArrayRead}\rvert)$	$(2^h-1)\cdot(\lvert\#\mathsf{NaiveRingLT}\rvert + \lvert\#\mathsf{NaiveRingEQ}\rvert + 12)$	$2^h\cdot(\lvert\#\mathsf{rArrayAND}\rvert + \lvert\#\mathsf{BitConversion}\rvert) + \lvert\#\mathsf{rInnerProduct}\rvert$
Naive Construction over Field	$(2^h-1)\cdot\lvert\#\mathsf{pArrayRead}\ \mathit{in\ off.}\rvert$	$(2^h-1)\cdot(\lvert\#\mathsf{pLTZ}\ \mathit{in\ off.}\rvert + \lvert\#\mathsf{pEQZ}\ \mathit{in\ off.}\rvert + \lvert\#\mathsf{pInnerProduct}\ \mathit{in\ off.}\rvert)$	$2^h\cdot\lvert\#\mathsf{pArrayOr}\ \mathit{in\ off.}\rvert + \lvert\#\mathsf{pInnerProduct}\ \mathit{in\ off.}\rvert$
(Protocol 2 in 4.2.12)	$(2^h-1)\cdot\lvert\#\mathsf{pArrayRead}\ \mathit{in\ on.}\rvert$	$(2^h-1)\cdot(\lvert\#\mathsf{pLTZ}\ \mathit{in\ on.}\rvert + \lvert\#\mathsf{pEQZ}\ \mathit{in\ on.}\rvert + \lvert\#\mathsf{pInnerProduct}\ \mathit{in\ on.}\rvert)$	$2^h\cdot\lvert\#\mathsf{pArrayOr}\ \mathit{in\ on.}\rvert + \lvert\#\mathsf{pInnerProduct}\ \mathit{in\ on.}\rvert$
[59]	0	0	0
	0	0	0
This Work over Ring (Protocol 13)	$(2^h-1)\cdot\lvert\#\mathsf{rFSelection}\ \mathit{in\ off.}\rvert$	$\lvert\#\mathsf{rComp}\ \mathit{in\ off.}\rvert$	$\lvert\#\mathsf{rPathEval}\ \mathit{in\ off.}\rvert$
	$(2^h-1)\cdot\lvert\#\mathsf{rFSelection}\ \mathit{in\ on.}\rvert$	$\lvert\#\mathsf{rComp}\ \mathit{in\ on.}\rvert$	$\lvert\#\mathsf{rPathEval}\ \mathit{in\ on.}\rvert$
This Work over Field (Protocol 14)	$(2^h-1)\cdot\lvert\#\mathsf{pFSelection}\ \mathit{in\ off.}\rvert$	$(2^h-1)\cdot(\lvert\#\mathsf{pLTZ}\ \mathit{in\ off.}\rvert + \lvert\#\mathsf{pEQZ}\ \mathit{in\ off.}\rvert + \lvert\#\mathsf{pInnerProduct}\ \mathit{in\ off.}\rvert$	$2^h\cdot\lvert\#\mathsf{pArrayOr}\ \mathit{in\ off.}\rvert + \lvert\#\mathsf{pInnerProduct}\ \mathit{in\ off.}\rvert$
	$(2^h-1)\cdot\lvert\#\mathsf{pFSelection}\ \mathit{in\ on.}\rvert$	$(2^h-1)\cdot(\lvert\#\mathsf{pLTZ}\ \mathit{in\ on.}\rvert + \lvert\#\mathsf{pEQZ}\ \mathit{in\ on.}\rvert + \lvert\#\mathsf{pInnerProduct}\ \mathit{in\ on.}\rvert$	$2^h\cdot\lvert\#\mathsf{pArrayOr}\ \mathit{in\ on.}\rvert + \lvert\#\mathsf{pInnerProduct}\ \mathit{in\ on.}\rvert$
This Work over Field (Protocol 15)	$(2^h-1)\cdot\lvert\#\mathsf{pFSelection}\ \mathit{in\ off.}\rvert$	$(2^h-1)\cdot(\lvert\#\mathsf{pLTZ}\ \mathit{in\ off.}\rvert + \lvert\#\mathsf{pEQZ}\ \mathit{in\ off.}\rvert + \lvert\#\mathsf{pInnerProduct}\ \mathit{in\ off.}\rvert$	$\lvert\#\mathsf{pPathEval}\ \mathit{in\ off.}\rvert$
	$(2^h-1)\cdot\lvert\#\mathsf{pFSelection}\ \mathit{in\ on.}\rvert$	$(2^h-1)\cdot(\lvert\#\mathsf{pLTZ}\ \mathit{in\ on.}\rvert + \lvert\#\mathsf{pEQZ}\ \mathit{in\ on.}\rvert + \lvert\#\mathsf{pInnerProduct}\ \mathit{in\ on.}\rvert$	$\lvert\#\mathsf{pPathEval}\ \mathit{in\ on.}\rvert$

4.1.3 Related Work

4.1.3.1 MSB Extraction Protocol.

The MSB extraction protocol extracts the (secret-shared) MSB from the (secret-shared) input while keeping the input and its MSB secret. It is known as a useful subprotocol for computing the mixed circuits (e.g., less-than protocol).

A semi-honest MSB extraction protocol with constant rounds over the field has been proposed [50]. However, the field-based MPC protocol has greater computational complexity than the ring-based one. Furthermore, the authors of [50] did not propose a maliciously secure construction with constant rounds. One of the existing MSB extraction protocols over the ring uses the GC to achieve constant rounds [10, 13, 71]. As another example, existing MSB extraction protocols over the ring use the circuit-based approach based only on SS-MPC [12, 14, 16–18][7]. In recent years, the constant-round MSB extraction protocol via only SS-MPC [22, 33, 70] has been proposed. However, this protocol achieves only semi-honest security.

[7]FLASH [14] and Trident [12] in the conference version include a constant-round MSB extraction protocol. However, a flaw was found in FLASH and was fixed in the preprint version of FLASH (uploaded to ePrint). The MSB extraction protocol of Trident used the same approach as FLASH and had the same flaw. The fixed MSB extraction protocols of [12, 14] need non-constant communication rounds depending on the size of modulus.

Table 4.6: Comparison of communication complexity of the oblivious shuffle protocols (Rounds: the number of communication rounds, Comm.: the number of (amortized) communication bits per all parties, n: the number of parties, t: the number of corruptions, ${}_nC_t$: the number of subset of t distinct elements of n parties, i.e., $n!/t!(n-t)!$, m: the length of array, p: prime number, $L(>1)$: arbitrary integer, $|(com+zk)\_round|$: the number of rounds of commitments and zero-knowledge proof, $|(com+zk)\_comm.|$: the number of communication bits of commitments and zero-knowledge proof., -: We consider fairness only against a malicious adversary, not semi-honest adversary.)

	Corruption, Security	Fairness	Rounds		Comm.	
			Offline	Online	Offline	Online
Resharing-based shuffle [67]	$t<n/2$, semi-honest	-	0	$2\cdot {}_nC_t$	0	${}_nC_t\cdot(t(n-t)$ $+(n-t)(n-1))\cdot m\log_2 L$
Resharing-based shuffle [67]	$t<n/2$, malicious	(abort)	0	$2\cdot {}_nC_t$ $+\|(com+zk)\_round\|$	0	${}_nC_t\cdot(t(n-t)+(n-t)(n-1))\cdot m\log_2 L$ $+\|(com+zk)\_comm.\|$
[68, 69]	$t<n/2$, semi-honest	-	0	6	0	$18m\log_2 p$
[62]	$t<n/2$, semi-honest	-	0	3	0	$6m\log_2 L$
Ours	$t<n/3$ malicious	✓	4	3	$9m\log_2 L$	$3m\log_2 L$

4.1.3.2 Feature Selection Protocol (Oblivious Array Read Protocol).

There are two types of OAR protocol: *circuit-based approach using general MPC* [61, 72–74] and *oblivious random access machine* (*ORAM*) [75–80]. Blanton et al. [60] proposed the OAR (and write) protocol with constant rounds. They demonstrated through their experiments that their scheme is superior to state-of-the-art schemes of the former [73] and the latter [75]. Therefore, to the best of our knowledge, the OAR protocols in [60] are the latest and most efficient protocol with constant rounds.

4.1.3.3 Comparison Protocol.

In [81–84], efficient comparison protocols are proposed. However, these schemes compute the less-than circuit and are not constant-rounds protocols. ABY [30] and ABY3 [10] can construct a constant-round comparison protocol over the ring by using GC. However, the constant-round comparison protocol using *only* SS-MPC over the ring is not proposed in [10, 30]. To the best of our knowledge, the SecureNN [33] is the only 3PC based on the secret sharing scheme (SS-3PC) including the MSB extraction protocol with constant rounds over the ring[8]. We note that SecureNN [33] did not propose a constant-round equality testing protocol.

[8]In the conference version of FLASH [14], Byali et al. proposed a constant-round MSB extraction protocol over the ring. However, the flaw was found and fixed in the preprint version uploaded to the ePrint server. The MSB extraction protocol of Trident [12] used the same approach as FLASH and had the same flaw. As a result, the MSB extraction protocols of FLASH and Trident are not constant-round protocols. Therefore, the fixed protocols of [12, 14] are not constant-round protocols and need many communication rounds depending on the size of modulus.

Table 4.7: Comparison of communication complexity of the MSB extraction protocols via (only) secret sharing over the ring (Rounds: the number of communication rounds, Comm.: the number of (amortized) communication bits per all parties, n: the number of parties, $t(=1)$: the number of corruptions, p': the smallest prime number larger than k, -: We consider fairness only against a malicious adversary, not semi-honest adversary.)

	Corruption, Security	Fairness	Rounds		Comm.	
			Offline	Online	Offline	Online
[22]	$t < n/2$, semi-honest	-	1	7	$6k^2 - 6k$	$11k + 3(k-1)\log_2 p + 4$
[70] (with pPIE)	$t < n/2$, semi-honest	-	0	2	0	$4(k+1)^2 + 4k$
[70] (with pPIE$'$)	$t < n/2$, semi-honest	-	0	3	0	$4(2k+1)(1+\log_2 k) + 4k$
ABY3 [10]+ [9]	$t < n/2$, malicious	(abort)	4	$1 + \log_2 k$	$24k$	$18k$
BLAZE [13]	$t < n/2$, malicious	✓	4	$1 + \log_2 k$	$9k$	$9k$
Trident [12]	$t < n/3$, malicious	✓	1	$1 + \log_2 k$	$3k$	$7k$
FLASH [14]	$t < n/3$, malicious	✓ (Robustness)	2	$3 + \log_2 k$	$4k$	$24k$
SWIFT [17]	$t < n/3$, malicious	✓ (Robustness)	1	$\log_2 k$	$7k - 6$	$7k - 6$
Fantastic Four [16]	$t < n/3$, malicious	✓ (Robustness)	0	$1 + \log_2 k$	0	$8k - 6$
Tetrad [18] (Tetrad-R^{Π})	$t < n/3$, malicious	✓ (Robustness)	1	$\log_4 k$	$3 \cdot 216 + 12 \cdot 184 + 33 \cdot 179 + k$	$3 \cdot (216 + 184 + 179) + k$
Ours	$t < n/3$, malicious	✓	17	9	$6k^2 + (69k - 57)\log_2 p' + 11k + 2$	$18(k-1)\log_2 p' + 15k$

4.1.3.4 Path Evaluation Protocol (k-ary AND/OR Protocol).

The path evaluation protocol outputs the shares of class labels assigned to the leaf node where the comparison result bits regarding the internal nodes included in its path are all 1. That is, it is easy to construct the path evaluation protocol if the k-ary AND/OR protocol exists.

Catrina and Hoogh [50] proposed a constant-round k-ary OR protocol over the field. Ohata and Nuida [84] proposed an efficient k-ary AND protocol (i.e., multi-fan-in multiplication/AND protocol) over the ring. However, their protocol is not a constant-round protocol. To the best of our knowledge, there is no constant-round k-ary AND protocol over the ring. Therefore, it is still non-trivial to construct a constant-round path evaluation protocol over the ring.

4.1.3.5 Private Decision Tree Evaluation Protocol.

There are many constructions of the PDTE protocol including those based on HE [85, 86] and GC+HE [57, 58]. Kiss et al. published [39] a systematization of knowledge paper that mainly focused on the constant-round constructions based on GC or HE. On the other hand, constructions of the PDTE protocol based on ORAM are proposed in [87, 88]. However, these constructions are not constant-round protocols.

Cock et al. [59] proposed an efficient PDTE protocol over the ring in commodity-based

Table 4.8: Comparison of communication complexity of the PDTE protocols via (only) secret sharing over the ring (Rounds: the number of communication rounds, Comm.: the number of (amortized) communication bits per all parties, n: the number of parties, $t(=1)$: the number of corruptions, m: number of features, k: bit length of ring, p': the smallest prime number greater than k, h: height of the tree)

	Corruption, Security	Rounds in Offline Rounds in Online	Comm. in Offline Comm. in Online
Protocol 9 in [22]	$t < n/2$, semi-honest	0	0
		$\log_2(m) + 2\log_2(k)$ $+\log_2(h) + 9$	$(2^h - 1) \cdot (3\log_2(m) + 3\log_2(m) \cdot \log_2(\log_2 m) + 6m \log_2 m$ $+4mk + 2m + 17k + 6k \log_2 k + 3) + 2^h \cdot (3h - 3 + 6k) + 3k$
[59]	$t < n/2$, semi-honest	0	0
		$h + \log_2(k) + 5$	$(2^h - 1) \cdot (10m \log_2(m) + 30k - 10\log_2(k) - 20) + 2^h \cdot 10hk$
Protocol 8 in [22]	$t < n/2$, semi-honest	1	$(2^h - 1) \cdot (\log_2(m) \cdot 6k^2 + 18k^2)$
		26	$(2^h - 1) \cdot \log_2(m) \cdot (9k + (3k - 3)\log_2 p' + 4)$ $+(2^h - 1) \cdot (27k + (9k - 9)\log_2 p' + 22) + 2^h(6k + 9h)$
Naive Construction (Protocol 3 in 4.2.13)	$t < n/3$ malicious	$\log_2 h + 11$	$2^h \cdot (30k + 3h) - 30k - 3$
		$\log_2 \log_2 m + \log_2 k$ $+\log_2 h + 9$	$2^h \cdot (17\log_2 m + 33k + 3h + 2) - 17\log_2 m - 33k - 4$
Ours	$t < n/3$, malicious	44	$(2^h - 1) \cdot ((12m + 18)k^2 + (138km + 207k - 114m$ $-171)\log_2 p' + 25mk + 7m + 25k + 12) + 9 \cdot 2^h \cdot (h + 1) \cdot (k + h)$
		27	$(2^h - 1) \cdot (36km - m54k - 54)\log_2 p' + 33mk$ $+4m + 48k + 6) + 3 \cdot 2^h \cdot (h + 1) \cdot (k + h) + 4 \cdot 2^h \cdot h$

two-party computation[9]. However, in their protocol, each party must know the features and threshold values or has the shares of the binary representation of these values. Furthermore, their protocol in [59] is not a constant-round protocol. Hence, to the best of our knowledge, there is no PDTE protocol with constant rounds over the ring using only SS-MPC.

4.1.3.6 Oblivious Shuffle Protocol.

The oblivious shuffle protocol shuffles the input (secret-shared) array while keeping the array elements and the shuffling order secret. It is known as a useful subprotocol for database operations. For example, oblivious sorting protocols [62, 68, 69] use it as a subprotocol.

The resharing-based shuffling protocol [67] is a typical oblivious shuffling protocol. In particular, in the case of semi-honest secure 3PC with single corruption, the resharing-based shuffling protocol can achieve constant rounds [62, 68, 69]. As a different direct approach, by using the MPC protocols for the arbitrary function and the permutation networks, the oblivious shuffling protocol can be achieved [89, 90]. However, this approach requires many rounds depending on the size of the permutation networks.

[9]The commodity-based two-party computation employs the client (i.e., helper entity) that generates the Beaver's multiplication triples and sends it to two parties in the offline phase. In the online phase, the client does not participate in the protocol or collude with any party.

We note that Tables 4.3 and 4.4 show the communication complexity of parties, not the client. Hence, if Tables 4.3 and 4.4 describe it associated with the parties and the client, the communication cost in the offline phase of [59] in Tables 4.3 and 4.4 is not 0 because there is the communication cost associated with the Beaver's multiplication triples.

We also note that Table 4.5 shows that the number of PRF invocations associated with parties, not the client. Hence, if Table 4.5 describes it associated with the parties and the client, the number of PRF invocations of [59] in Table 5 is not 0 because the client needs to invoke the PRF to generate the Beaver's multiplication triples.

4.2 Preliminaries in Chapter 4

4.2.1 2-out-of-3 Replicated Secret Sharing Scheme ((2,3)-RSS) and 2-out-of-2 Additive Secret Sharing Scheme ((2,2)-ASS)

We denote the (2,3)-RSS shares of x on $\mathbb{Z}_L$ ($L = 2,\ 2^k,\ p,\ p'$) by $[x]_L = ([x]_{L,0},\ [x]_{L,1},\ [x]_{L,2})$ where $x \in \mathbb{Z}_L$. P_i has the share of x, $[x]_{L,i} = (x_i,\ x_{i+1})$ where $x = x_0 + x_1 + x_2 \mod L$ ($x_i \in \mathbb{Z}_L,\ i = 0, 1, 2$) and $x_{2+1} = x_0$. We also denote the (2,2)-ASS shares of x on $\mathbb{Z}_L$ by $\langle x \rangle_{L,(i,j)} = (\langle x \rangle_{L,i}, \langle x \rangle_{L,j})$. P_i and P_j have the share of x, $\langle x \rangle_{L,i} = x_i$ and $\langle x \rangle_{L,j} = x_j$ where $x = x_i + x_j \mod L$ ($x_i \in \mathbb{Z}_L,\ i, j \in \{0, 1, 2\}$ ($i \neq j$)), respectively. In particular, we assume $0 \leq x \leq 2^{k-1} - 1$ if we use $[x]_{2^k}$ or $\langle x \rangle_{2^k}$.

4.2.2 2-out-of-3 Shamir's Secret Sharing Scheme ($(2,3)$-SSS)

We employ the Shamir's secret sharing scheme [91]. We denote the $(2,3)$-SSS share of x on $\mathbb{F}_p$ by $[\![x]\!]_p = ([\![x]\!]_{p,0},\ [\![x]\!]_{p,1},\ [\![x]\!]_{p,2})$. P_i has the share of x, $[\![x]\!]_{p,i}$.

4.2.3 2-out-of-4 Replicated Secret Sharing Scheme ($(2,4)$-RSS)

We use the $(2,4)$-RSS in [12][10]. We denote the $(2,4)$-RSS's shares of x on $\mathbb{Z}_L$ as $[x]_L$. P_3 has the share $[x]_{L,3} = (\lambda_{x,0},\ \lambda_{x,1},\ \lambda_{x,2})$. $P_{i'}$ ($i' \in \{0, 1, 2\}$) has the share $[x]_{L,i'} = (\mathsf{m}_x,\ \lambda_{x,\overline{i'+1}},\ \lambda_{x,\overline{i'-1}})$, where $\lambda_{x,\overline{2+1}} = \lambda_{x,0}$. It holds that $\mathsf{m}_x = x + \lambda_x \bmod L$ and $\lambda_x = \lambda_{x,0} + \lambda_{x,1} + \lambda_{x,2} \bmod L$ where $\mathsf{m}_x,\ \lambda_x,\ \lambda_{x,0},\ \lambda_{x,1},\ \lambda_{x,2} \in \mathbb{Z}_L$. Note that m_x is computed depending on an actual input x while $\lambda_{x,i'}$ ($i' = 0, 1, 2$) can be generated independently of x by the pseudo-random function.

4.2.4 Secure Three-party Computation with One Corruption over Ring

We use the same addition and multiplication of shares as [5] denoted as $[x]_L + [y]_L$ and $[x]_L \cdot [y]_L$, respectively. We also use the same scalar addition and multiplication of shares as [5] denoted as $c + [x]_L$ and $c \cdot [x]_L$ where $c \in \mathbb{Z}_L$, respectively. We use the same notation for operations of scalars and one of the shares to keep the description simple.

We also note that P_i has the pair of shared keys $(\mathsf{k}_i, \mathsf{k}_{i+1})$ where $\mathsf{k}_i \in \{0, 1\}^\kappa$ and $\mathsf{k}_{2+1} = \mathsf{k}_0$.

4.2.5 Secure Three-party Computation with One Corruption over Field

We employ the same addition and multiplication of shares and same data format as ones used in [50]. We use the same notation for operations of scalars and one of the shares as in the 3PC over the ring to keep the description simple. We also note that P_i has the pair of shared keys in the same way as the 3PC over the ring.

[10]Note that the subscripts of the parties have been reassigned: P_0, P_1, P_2, P_3 in [12] correspond to P_3, P_0, P_1, P_2 in this chapter.

4.2.6 Fair Four-party Computation with One Corruption over Ring

Each party has pre-shared keys in the same way as [12]. That is, each pair of P_i and P_j (where $i \neq j$ and $i, j \in \{0, 1, 2, 3\}$) has $\mathsf{k}_{i,j} \in \{0,1\}^\kappa$. Each group of P_a, P_b and P_c (where $a \neq b \neq c$ and $a, b, c \in \{0, 1, 2, 3\}$) has $\mathsf{k}_{a,b,c} \in \{0,1\}^\kappa$. All parties have $\mathsf{k} \in \{0,1\}^\kappa$. We assume that each party is connected by a point-to-point private and authenticated channel in the same way as [12].

We use the same addition and multiplication of shares as [12] and denote them by $[x]_L + [y]_L$ and $[x]_L \cdot [y]_L$, respectively. We also use the same scalar addition and multiplication of shares as [12] and denote them as $c + [x]_L$ and $c \cdot [x]_L$ where $c \in \mathbb{Z}_L$, respectively. In this chapter, the same notation is used for scalar operations and share operations to simplify the description.

4.2.7 Building Blocks of Three-party Computation Protocol over Ring

The following building blocks are secure with computational indistinguishability in the presence of one semi-honest corrupted party [5, 10, 33, 60, 62, 67]. These protocols are universal composability (UC) secure [92], so they can be combined with other protocols.

- $[x]_{2^k} \leftarrow \mathsf{BitConversion}([x]_2)$: It takes $[x]_2$ (where $x \in \mathbb{Z}_2$) as inputs and outputs $[x]_{2^k}$. For more details, see [10]. It requires 1 round and $6k$ bits as its communication cost. It invokes the PRF 14 times.

- $[r]_{L,i} \leftarrow \mathsf{RndGen}(L, \mathsf{k}_i, \mathsf{k}_{i+1})$: It generates the share of a random value on $\mathbb{Z}_L$. It is called by each P_i using the PRF F_L and the pair of shared keys $(\mathsf{k}_i, \mathsf{k}_{i+1})$. For more details, see [5]. It requires no communication cost. It invokes the PRF 6 times.

- $[x]_L \leftarrow \mathsf{rShare}(L, P_i, x)$: It takes the modulus L, the input dealer P_i and the input value x as inputs and outputs $[x]_L$. For more details, see [5]. It requires 1 round and $4\log_2(L)$ bits as its communication cost. It invokes the PRF twice.

- $x \leftarrow \mathsf{rOpen}(P_i, [x]_{2^k})$: It takes the receiver P_i and the share $[x]_{2^k}$ as inputs and outputs x. For more details, see [5]. It requires 1 round and $\log_2(L)$ bits as its communication cost. It does not invoke the PRF.

- $\langle x \rangle_{L,(i,i+1)} \leftarrow \mathsf{aShare}(P_{i+2}, L, x, P_i, P_{i+1})$: It takes the input dealer P_{i+2}, the modulus L, the input value x and the receivers P_i and P_{i+1} as inputs and outputs $\langle x \rangle_{L,(i,i+1)}$. It requires 1 round and $2\log_2(L)$ bits as its communication cost. It invokes the PRF once.

- $[x_a]_{2^k} \leftarrow \mathsf{rArrayRead}(\{[x_j]_{2^k}\}_{j=0}^{m-1}, [a]_m)$: It takes the shared array $\{[x_j]_{2^k}\}_{j=0}^{m-1}$ and the shared index $[a]_m$ as inputs and outputs $[x_a]_{2^k}$. First, P_0 and P_1 generate $\{\langle x_j \rangle_{2^k,(0,1)}\}_{j=0}^{m-1}$ from $\{[x_j]_{2^k}\}_{j=0}^{m-1}$ by setting $\langle x_j \rangle_{2^k,0} = x_{j,0} + x_{j,1} \mod 2^k$ for P_0 and $\langle x_j \rangle_{2^k,1} = x_{j,2}$ for P_1 where $[x_j]_{2^k,0} = (x_{j,0}, x_{j,1})$ and $[x_j]_{2^k,1} = (x_{j,1}, x_{j,2})$. Each party generates the $(2,2)$-ASS share of an index from $[a]_m$. Then, each party runs the OAR protocol of the custom three-party construction [60] and obtains $\langle x_a \rangle_{2^k,(0,2)}$. Next, P_0 and P_2 run $[\langle x_a \rangle_{2^k,0}]_{2^k} \leftarrow \mathsf{rShare}(2^k, P_0, \langle x_a \rangle_{2^k,0})$ and $[\langle x_a \rangle_{2^k,2}]_{2^k} \leftarrow \mathsf{rShare}(2^k, P_2, \langle x_a \rangle_{2^k,2})$,

respectively. Finally, it outputs $[x_a]_{2^k} = [\langle x_a \rangle_{2^k,0}]_{2^k} + [\langle x_a \rangle_{2^k,2}]_{2^k}$. It requires 3 rounds and $4mk + 2m + 8k$ bits as its communication cost. It invokes the PRF $4m + 4$ times.

- $b \leftarrow \mathsf{PrivateCompare}(x, \{\langle a|_j \rangle_{p',(0,1)}\}_{j=0}^{k-1}, \beta, \{s_j\}_{j=0}^{k-1}, \{u_j\}_{j=0}^{k-1}, P_2)$: This is the private compare protocol [33]. It takes the value $x (\in \mathbb{Z}_{2^k})$ known to P_0 and P_1, the shares of binary values $\{\langle a|_j \rangle_{p',(0,1)}\}_{j=0}^{k-1}$ $(a|_j \in \mathbb{Z}_2)$, the random bit (known to P_0 and P_1) β, the random values (known to P_0 and P_1) $\{s_j\}_{j=0}^{k-1}$ and $\{u_j\}_{j=0}^{k-1}$ (where $s_j, u_j \in \mathbb{F}_{p'}^*$) and the receiver P_2 as inputs and outputs the masked comparison result bit $b = \beta \oplus (a > x)$ to P_2. It takes 1 round and $2k \log_2(p')$ bits as its communication cost. It invokes the PRF $2k + 1$ times.

- $\{\mathcal{R}'_j\}_{j=0}^{m-1} \leftarrow \mathsf{rSetShuffle}(\{\mathcal{R}_j\}_{j=0}^{m-1})$: It is the oblivious shuffle protocol for the multidimensional shared array. Let $\mathcal{R}_j = \{[v_j]_{2^k}, [c_{j,0}]_2, \ldots, [c_{j,h-1}]_2\}$ be the set of shares and S_m be the set of all permutation $\sigma : \{0, \ldots, m-1\} \to \{0, \ldots, m-1\}$. It takes the set of shares $\{\mathcal{R}_j\}_{j=0}^{m-1}$ as inputs and outputs $\{\mathcal{R}'_j\}_{j=0}^{m-1} = \{[v'_j]_{2^k}, [c'_{j,0}]_2, \ldots, [c'_{j,h-1}]_2\}$ such that $v'_{\sigma(j)} = v_j$, $c'_{\sigma(j),\ell} = c_{j,\ell}$ (for $\ell = 0, \ldots, h-1$) and $\mathcal{R}'_{\sigma(j)} = \mathcal{R}_j$ while no one knows σ. Intuitively, it runs the oblivious shuffle protocol for (single) shared array [62,67] in parallel. It requires 3 rounds and $6m(k+h)$ bits as its communication cost. It invokes the PRF $6mh$ times.

- $[\bigwedge_{j=0}^{h-1} b_j]_2 \leftarrow \mathsf{rArrayAND}(\{[b_j]_2\}_{j=0}^{h-1})$: It takes $\{[b_j]_2\}_{j=0}^{h-1}$ as inputs and outputs $[\prod_{j=0}^{h-1} b_j]_2$. Intuitively, it computes $h-1$ AND gates by using secure multiplication [5]. It requires $\log_2(h)$ rounds and $3(h-1)$ bits as its communication cost. It invokes the PRF $6(h-1)$ times.

- $\{[x|_j]_2\}_{j=0}^{m-1} \leftarrow \mathsf{NaiveBitDec}(m, [x]_{2^k})$: It takes $[x]_{2^k}$ and $m (\leq k)$ as inputs and outputs $\{[x|_j]_2\}_{j=0}^{m-1}$. For more details, see [10]. It requires $1 + \log_2(m)$ rounds and $3m + 3\log_2(m)$ bits as its communication cost. It invokes the PRF $2 + 6\log_2(m)$ times.

- $[b]_2 \leftarrow \mathsf{NaiveRingLT}([x]_{2^k}, [y]_{2^k})$: It takes $[x]_{2^k}$ and $[y]_{2^k}$ as inputs and outputs $[b]_2$ (where $b = 1$ iff $x < y$ and $b = 0$ otherwise). Intuitively, it runs $\{[(x-y)|_j]_2\}_{j=0}^{k-1} \leftarrow \mathsf{NaiveBitDec}(k, [x-y]_{2^k})$ internally. Then, it outputs $[b]_2 = [\mathsf{msb}(x-y)]_2$. It requires $1 + \log_2(k)$ rounds and $3k + 3\log_2(k)$ bits as its communication cost. It invokes the PRF $2 + 6\log_2(k)$ times.

- $[b]_2 \leftarrow \mathsf{NaiveRingEQ}([x]_{2^k}, [y]_{2^k})$: It takes $[x]_{2^k}$ and $[y]_{2^k}$ as inputs and outputs $[b]_2$ (where $b = 1$ iff $x = y$ and $b = 0$ otherwise). Intuitively, it runs $\{[(x-y)|_j]_2\}_{j=0}^{k-1} \leftarrow \mathsf{NaiveBitDec}(k, [x-y]_{2^k})$ internally. Then, it runs $[b]_2 \leftarrow \mathsf{rArrayAND}(\{[(x-y)|_j]_2 \oplus 1\}_{j=0}^{k-1})$ and outputs $[b]_2$. It requires $1 + 2\log_2(k)$ rounds and $6k + 3k\log_2(k) - 3$ bits as its communication cost. It invokes the PRF $2 + 6\log_2(k) + 6k - 6$ times.

- $[\sum_{j=0}^{h-1} x_j \cdot y_j]_{2^k} \leftarrow \mathsf{rInnerProduct}(\{[x_j]_{2^k}\}_{j=0}^{h-1}, \{[y_j]_{2^k}\}_{j=0}^{h-1})$: It takes $\{[x_j]_{2^k}\}_{j=0}^{h-1}$ and $\{[y_j]_{2^k}\}_{j=0}^{h-1}$ as inputs and outputs $[\sum_{j=0}^{h-1} x_j \cdot y_j]_{2^k}$. For more details, see [5]. It requires 1 round and $3k$ bits as its communication cost. It invokes the PRF 6 times.

4.2.8 Building Blocks of Three-party Computation Protocol over Field

The following building blocks are secure with computational indistinguishability in the presence of one semi-honest corrupted party [2,50,60,67,69,91,93]. These protocols are UC secure [92], so it can be combined with the other protocols. We note that one invocation of a primitive during which all parties send their shares (i.e., the field elements) to each other requires 1 round and $6\log_2(p)$ bits as communication cost.

- $[\![\bigvee_{j=0}^{h-1} b_j]\!]_p \leftarrow \mathsf{pArrayOr}(\{[\![b_j]\!]_p\}_{j=0}^{h-1})$: It is the logical sum protocol of h bits [50]. It takes $\{[\![b_j]\!]_p\}_{j=0}^{h-1}$ ($b_j \in \{0,1\}$) as inputs and outputs $[\![\bigvee_{j=0}^{h-1} b_j]\!]_p$. It requires 1 round and $\log_2(h) \cdot 6\log_2(p)$ bits in the offline phase and 3 rounds and $(3\log_2(h)) \cdot 6\log_2(p)$ bits in the online phase as its communication cost. It invokes the PRF $\log_2(h)+1$ times in the offline phase and $6\log_2(h)$ times in the online phase.

- $[\![b]\!]_p \leftarrow \mathsf{pLTZ}(k, [\![a]\!]_p)$: It is the less-than-zero protocol [50]. It takes the bit length of value k (s.t. $2^k < p$) and the share $[\![a]\!]_p$ (s.t. $-2^{k-1} \leq a \leq 2^{k-1} < p$) as inputs and outputs $[\![b]\!]_p$ (s.t. $b = 1$ iff $a < 0$ and $b = 0$ otherwise). It requires the 1 round and $(3k-3) \cdot 6\log_2(p)$ bits in the offline phase and 3 rounds and $(k+1) \cdot 6\log_2(p)$ bits in the online phase as its communication cost. It invokes the PRF $19\log_2(p)$ times in the offline phase and $6\log_2(p)$ times in the online phase.

- $[\![b]\!]_p \leftarrow \mathsf{pEQZ}(k, [\![a]\!]_p)$: It is the equality testing protocol [50]. It takes the bit length of value k (s.t. $2^k < p$) and the share $[\![a]\!]_p$ (s.t. $-2^{k-1} \leq a \leq 2^{k-1} < p$) as inputs and outputs $[\![b]\!]_p$ (s.t. $b = 1$ iff $a = 0$ and $b = 0$ otherwise). It requires the 1 round and $(k - 3\log_2(k) - 2) \cdot 6\log_2(p)$ bits in the offline phase and 3 rounds and $(\log_2(k)+2) \cdot 6\log_2(p)$ bits in the online phase as its communication cost. It invokes the PRF $2k + 24\log_2(k) + 2$ times in the offline phase and $6\log_2(k)+1$ times in the online phase.

- $[\![\sum_{j=0}^{m-1} x_j \cdot y_j]\!]_p \leftarrow \mathsf{pInnerProduct}(\{[\![x_j]\!]_p\}_{j=0}^{m-1}, \{[\![y_j]\!]_p\}_{j=0}^{m-1})$: It is the inner-product protocol [50,93]. It takes the shares $\{[\![x_j]\!]_p\}_{j=0}^{m-1}$ and $\{[\![y_j]\!]_p\}_{j=0}^{m-1}$ as inputs and outputs $[\![\sum_{j=0}^{m-1} x_j \cdot y_j]\!]$. It requires 1 round and $6\log_2(p)$ bits as its communication bits. It invokes the PRF 6 times in the online phase.

- $[\![x]\!]_p \leftarrow \mathsf{pShare}(P_i, x)$: It is the input sharing protocol by using $(2,3)$-SSS. It takes the input dealer P_i and the input value x as inputs and outputs $[\![x]\!]_p$. For more details, see [2,91]. It requires 1 round and $2\log_2(p)$ bits as communication cost. It invokes the PRF twice in the online phase.

- $x \leftarrow \mathsf{pOpen}(P_i, [\![x]\!]_p)$: It is the opening protocol by using $(2,3)$-SSS. It takes the receiver P_i and the share $[\![x]\!]_p$ as inputs and outputs x. For more details, see [2,91]. It requires 1 round and $\log_2(p)$ bits as its communication cost. It does not invoke the PRF.

- $\{\mathcal{R}'_j\}_{j=0}^{m-1} \leftarrow \mathsf{pSetShuffle}(\{\mathcal{R}_j\}_{j=0}^{m-1})$: It is the oblivious shuffle protocol for multidimensional shared array. Let $\mathcal{R}_j = \{[\![v_j]\!]_p, [\![c]\!]_p\}$ be the set of shares. Let S_m be the

set of all permutation $\sigma : \{0, \ldots, m-1\} \to \{0, \ldots, m-1\}$. It takes the set of shares $\{\mathcal{R}_j\}_{j=0}^{m-1}$ as inputs and outputs $\{\mathcal{R}'_j\}_{j=0}^{m-1} = \{[\![v'_j]\!]_p, [\![c'_j]\!]_p\}$ such that $v'_{\sigma(j)} = v_j$, $c'_{\sigma(j)} = c_j$ and $\mathcal{R}'_{\sigma(j)} = \mathcal{R}_j$ while no one knows σ. Intuitively, it runs the oblivious shuffle protocol for (single) shared array [67, 69] in parallel. It requires 6 rounds and $18m \log_2(p)$ bits as its communication cost. It invokes the PRF $9m$ times in the online phase.

- $[\![x_a]\!]_p \leftarrow \mathsf{pArrayRead}(\{[\![x_j]\!]_p\}_{j=0}^{m-1}, [\![a]\!]_p)$: It takes the shared array $\{[\![x_j]\!]_p\}_{j=0}^{m-1}$ and the shared index $[\![a]\!]_p$ as inputs and outputs $[\![x_a]\!]_p$. First, P_0 and P_1 generate $\{\langle x_j \rangle_{p,(0,1)}\}_{j=0}^{m-1}$ from $\{[\![x_j]\!]_p\}_{j=0}^{m-1}$ by running the converting protocol ($\mathsf{S2A}$ in [60]) to convert the share of $(2,3)$-SSS to one of $(2,2)$-ASS without communication. Then, each party runs the OAR protocol of custom three-party construction [60] and gets $\langle x_a \rangle_{p,(0,2)}$. We employ the Mod protocol [50] to perform the modulo operation on $[\![a]\!]_p$ with m in the same way as [60]. In the custom three-party construction [60], it uses the oblivious random rotation by using the random number r_1 and r_2 and computes $[\![a+r_1+r_2]\!]_p$. We require the Mod protocol [50] to know the rotated target index, i.e., $a+r_1+r_2 \mod m$. Next, P_0 and P_2 run $[\![\langle x_a \rangle_{p,0}]\!]_p \leftarrow \mathsf{pShare}(P_0, \langle x_a \rangle_{p,0})$ and $[\![\langle x_a \rangle_{p,2}]\!]_p \leftarrow \mathsf{pShare}(P_2, \langle x_a \rangle_{p,2})$, respectively. Finally, it outputs $[\![x_a]\!]_p = [\![\langle x_a \rangle_{p,0}]\!]_p + [\![\langle x_a \rangle_{p,2}]\!]_p$. That is, P_0 and P_2 run the converting protocol ($\mathsf{A2S}$ in [60]) from the share of $(2,2)$-ASS to one of $(2,3)$-SSS. It requires 1 round and $(6\log_2(m) - 2) \cdot \log_2(p)$ bits in the offline phase and 10 rounds and $(4m + 2\log_2(m) + 12) \cdot \log_2(p)$ bits in the online phase as its communication cost. It invokes the PRF $18\log_2(p) + 12m$ times in the offline phase and $4m + 12\log_2(p) + 5$ times in the online phase.

4.2.9 Building Blocks of Fair Four-party Computation Protocol over Ring

We use the following building blocks of Trident [12] in our protocol as subprotocols.

- $v \leftarrow \mathsf{CC}(\{P_a, P_b\}, v, P_c)$: It runs the cross-checking message transfer protocol (also known as the joint message passing protocol [16]). It takes the senders $\{P_a, P_b\}$, $v \in \mathbb{Z}_L$ that the senders have and the receiver P_c, where $(a, b, c \in \{0, \ldots, 3\}$ and $a \neq b \neq c)$. In CC, both senders send v to the receiver. Then, the receiver checks whether the values sent by the two senders match or not. If the values match, the receiver gets the correct value v and broadcasts the message $\mathsf{continue}$. If not, the receiver broadcasts the message abort and aborts the protocol. It takes 1 round and $\log_2 L$ bits as (amortized) communication cost. This cross-checking message transfer protocol is commonly used by recent fair 4PC protocols [12, 15, 16].

- $x \leftarrow \mathsf{OpenOne}(P_i, [x]_L)$: It runs the opening protocol for one party. It takes the receiver P_i and the share $[x]_L$ and outputs $x \in \mathbb{Z}_L$ to P_i. It requires 1 round and $\log_2(L)$ bits as its (amortized) communication cost.

- $[\sum_{j=0}^{M-1} x_j \cdot y_j]_L \leftarrow \mathsf{DotProd}(([x_0]_L, \ldots, [x_{M-1}]_L), ([y_0]_L, \ldots, [y_{M-1}]_L))$: It runs the dot product protocol. It takes the vectors of shares, $([x_0]_L, \ldots, [x_{M-1}]_L)$ and $([y_0]_L, \ldots, [y_{M-1}]_L)$ and then outputs $[\sum_{j=0}^{M-1} x_j \cdot y_j]_L$. It requires 1 round and $3\log_2(L)$ bits

as its (amortized) communication cost in the offline phase and 1 round and $3\log_2(L)$ bits as its (amortized) communication cost in the online phase. Note that we use Mult instead of $\mathsf{DotProd}$ when $M = 1$.

- $[x]_L \leftarrow \mathsf{BitConv}(L, [x]_2)$: It runs the bit conversion protocol (also known as Π_{Bit2A}, bit to arithmetic sharing protocol in [12][11]). It takes $[x]_2$ (where $x \in \mathbb{Z}_2$) and outputs $[x]_L$. It requires 2 rounds and $3\log_2(L)+1$ bits as its (amortized) communication cost in the offline phase and 1 round and $3\log_2(L)$ bits as its (amortized) communication cost in the online phase.

- $[a < b]_2 \leftarrow \mathsf{NaiveLT}([a]_{2^k}, [b]_{2^k})$: It takes the shares $[a]_{2^k}$ and $[b]_{2^k}$ and outputs the shares of the result of LT $[a < b]_2$. It can be achieved by using the revised (i.e., non constant-round) MSB extraction of [12] in Π_{LT} instead of Π_{msbExt}. $\mathsf{NaiveLT}$ needs 1 round and $3k$ bits in the offline phase and $1 + \log_2 k$ rounds and $7k$ bits in the online phase as (amortized) communication cost.

- $[a == b]_2 \leftarrow \mathsf{NaiveEQ}([a]_{2^k}, [b]_{2^k})$: It takes the shares $[a]_{2^k}$ and $[b]_{2^k}$ and outputs the shares of the result of EQ $[a == b]_2$. It can be achieved by using $\mathsf{NaiveLT}$ in Π_{EQ} instead of Π_{LT}. $\mathsf{NaiveEQ}$ needs 2 rounds and $9k$ bits in the offline phase and $2 + \log_2 k$ rounds and $17k$ bits in the online phase as (amortized) communication cost.

- $[\bigwedge_{j=0}^{h-1} b_j]_2 \leftarrow \mathsf{NaiveArrayAND}(\{[b_j]_2\}_{j=0}^{h-1})$: It takes $\{[b_j]_2\}_{j=0}^{h-1}$ and outputs $[\bigwedge_{j=0}^{h-1} b_j]_2$. Intuitively, it computes $h-1$ AND gates by using secure multiplication [12]. It needs $\log_2 h$ rounds and $3(h-1)$ bits in the offline phase and $\log_2 h$ rounds and $3(h-1)$ bits in the online phase as (amortized) communication cost.

Table 4.9 shows the communication complexities for the building blocks.

4.2.10 Structure of Decision Tree

We use the same structure as [22]. We denote an input array as an m-dimension feature vector by $\{\mathsf{attr}_i\}_{i=0}^{m-1}$ s.t. $0 \le \mathsf{attr}_i \le 2^{k-1}-1$. Let attr_i be the i-th feature. We assume that a decision tree is a complete binary tree. We define the decision tree as $\mathcal{T} = (h, \delta, \{\mathsf{idx}_j\}_{j=0}^{2^h-2}, \{\mathsf{v}_j\}_{j=0}^{2^h-2}, \{\mathsf{cond}_j\}_{j=0}^{2^h-2}, \{\mathsf{leafVal}_{j'}\}_{j'=0}^{2^h-1})$ where h is the height of the tree. Let $\{\mathsf{idx}_j\}_{j=0}^{2^h-2}$ be a set of the index values $\mathsf{idx}_j \in \mathbb{Z}_m$ s.t. $0 \le m \le 2^{k-1}-1$. The $j(=0,\ldots,2^h-2)$-th index value idx_j is used to choose the idx_j-th feature for comparison at the j-th internal node. We denote a set of the decision threshold values (assigned to the j-th internal node) by $\{\mathsf{v}_j\}_{j=0}^{2^h-2}$ s.t. $0 \le \mathsf{v}_j \le 2^{k-1}-1$. We also denote the set of the conditional bits to choose comparison operations (less-than (LT, $<$) or equality-testing (EQ, $==$)) by $\{\mathsf{cond}_j\}_{j=0}^{2^h-2}$. In each internal node, we use LT operation as a comparison operation and check whether $\mathsf{attr}_{\mathsf{idx}_j} < \mathsf{v}_j$ if $\mathsf{cond}_j = 1$. If not, we use the EQ operation and check whether $\mathsf{attr}_{\mathsf{idx}_j} == \mathsf{v}_j$.

[11] In [12], $\mathsf{BitConv}$ converts the shares on $\mathbb{Z}_2$ into the shares on $\mathbb{Z}_{2^k}$. It can be generalized to convert the shares on $\mathbb{Z}_2$ into the shares on $\mathbb{Z}_L$ (including $\mathbb{Z}_p$) by modifying $\mathsf{u}, \mathsf{v}, \mathsf{r}'_b$, and x' on $\mathbb{Z}_{2^k}$ to ones on $\mathbb{Z}_L$, since Trident [12] can work on an arbitrary ring.

Table 4.9: Communication complexity of the building blocks (Rounds: the number of communication rounds, Comm.: the number of (amortized) communication bits per all parties, L: modulus size, k: bit length of power-of-two ring, h: number of bits)

	Rounds		Comm.	
	Offline	Online	Offline	Online
CC	0	1	0	$\log_2 L$
OpenOne	0	1	0	$\log_2 L$
Mult	1	1	$3\log_2 L$	$3\log_2 L$
DotProd	1	1	$3\log_2 L$	$3\log_2 L$
BitConv	2	1	$3\log_2(L)+1$	$3\log_2 L$
NaiveLT	1	$1+\log_2 k$	$3k$	$7k$
NaiveEQ	2	$2+\log_2 k$	$9k$	$17k$
NaiveArrayAND	$\log_2 h$	$\log_2 h$	$3(h-1)$	$3(h-1)$

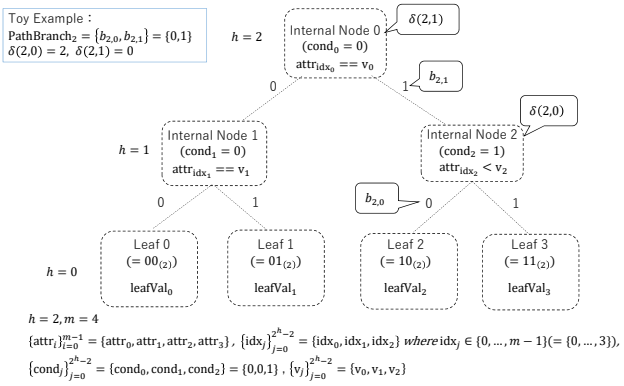

Figure 4.1: Toy example of decision tree structure (Copyright(C)2022 IEICE, [1] Fig.1)

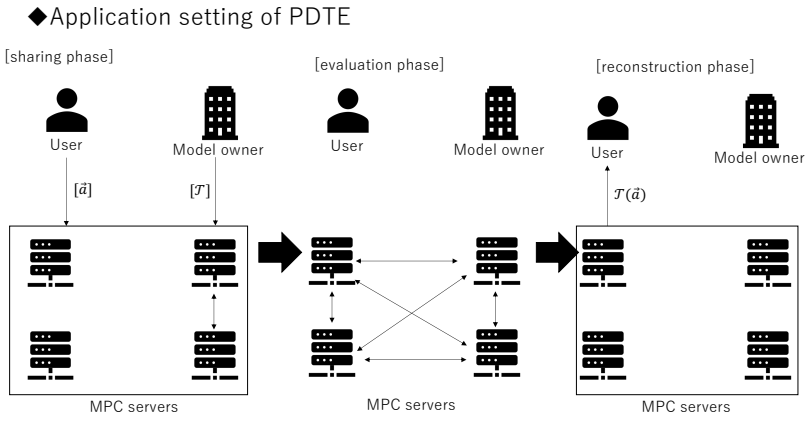

Figure 4.2: Overview of application setting for PDTE

We assign a comparison result bit to branches. That is, we assign the comparison result bit 1 (i.e., true) and 0 (i.e., false) to the right and left branch, respectively. The next step is to judge the right (or left) child node if the comparison result bit is 1 (or 0).

Each $j'(\in \{0, 1, \ldots, 2^h - 1\})$-th leaf node has the class label value $\mathsf{leafVal}_{j'} \in \mathbb{Z}_{2^k}$. We denote a set of class label values assigned to leaf nodes by $\{\mathsf{leafVal}_{j'}\}_{j'=0}^{2^h-1}$. We also denote a set of paths to the leaf nodes by $\mathsf{PathBranch}_{j'} = \{b_{j',\ell}\}_{\ell=0}^{h-1}$ (where $b_{j',\ell} \in \{0, 1\}$). Let $b_{j',\ell}$ be the bit assigned to the branch at the height ℓ in the path to the j'-th leaf node. Let also $\delta : \{0, \ldots, 2^h - 1\} \times \{0, \ldots, h-1\} \to \{0, \ldots, 2^h - 2\}$ be the map function that takes j' (i.e., the index number of the leaf node) and the height ℓ and outputs the position of the corresponding internal node. Fig. 4.1 shows an example of the same tree used in [22].

4.2.11 Application Setting

Fig. 4.2 shows the overview of the application setting for the PDTE.

Loosely speaking, an user distributes his/her attribute vector, $\vec{a}$ among MPC servers. A model owner also distributes his/her tree model, $\mathcal{T}$ among MPC servers. Let $[\vec{a}]$ and $[\mathcal{T}]$ be the shares of $\vec{a}$ and $\mathcal{T}$, respectively. Then, MPC servers compute the shares of the result, $[\mathcal{T}(\vec{a})]$ and send it to the user. The user reconstructs $\mathcal{T}(\vec{a})$ from received shares.

4.2.12 Naive Construction of PDTE with Semi-honest Security

To the best of our knowledge, Protocols 1 and 2 are the naive constructions, i.e., the best combination of the existing protocols based only on SS-3PC over the ring and field.

Protocol 1 Naive Construction of PDTE via 3PC over Ring

Input: Set of shared index number $\{[\mathsf{idx}_j]_{2^k}\}_{j=0}^{2^h-2}$, shared feature array $\{[\mathsf{attr}_i]_{2^k}\}_{i=0}^{m-1}$, threshold values $\{[\mathsf{v}_j]_{2^k}\}_{j=0}^{2^h-2}$, conditional values $\{[\mathsf{cond}_j]_{2^k}\}_{j=0}^{2^h-2}$, set of shared values assigned to leaf nodes $\{[\mathsf{leafVal}_{j'}]_{2^k}\}_{j'=0}^{2^h-1}$, mapping function δ

Output: shared values assigned to leaf node of correct path $[\mathsf{leafVal}_{j'}]_{2^k}$ where j' is s.t. $\bigwedge_{\ell=0}^{h-1}(j'|_\ell \oplus \mathsf{comp}_{\delta(j',\ell)} \oplus 1) = 1$. Let $\mathsf{comp}_{\delta(j',\ell)}$ be $\mathsf{cond}_{\delta(j',\ell)} \cdot (\mathsf{v}_{\delta(j',\ell)} < \mathsf{attr}_{\mathsf{idx}_{\delta(j',\ell)}}) + (1 - \mathsf{cond}_{\delta(j',\ell)}) \cdot (\mathsf{v}_{\delta(j',\ell)} == \mathsf{attr}_{\mathsf{idx}_{\delta(j',\ell)}})$.

1: **for** $j = 0, \ldots, 2^h - 2$ **do in parallel**
2: (Feature Selection Phase)
3: $\{[\mathsf{idx}_j|_{i'}]_2\}_{i'=0}^{\log_2(m)-1} \leftarrow \mathsf{NaiveBitDec}(\log_2(m), [\mathsf{idx}_j]_{2^k})$ // $1 + \log_2(\log_2 m)$ rounds & $3\log_2(m) + 3\log_2(\log_2(m))$ bits
4: **for** $i' = 0, \ldots, \log_2(m) - 1$ **do in parallel**
5: $[\mathsf{idx}_j|_{i'}]_m \leftarrow \mathsf{BitConversion}(m, [\mathsf{idx}_j|_{i'}]_2)$ // 1 round & $6m$ bits
6: **end for**
7: $[\mathsf{idx}_j]_m = \sum_{i'=0}^{\log_2(m)} 2^{i'} \cdot [\mathsf{idx}_j|_{i'}]_m$
8: $[\mathsf{attr}_{\mathsf{idx}_j}]_{2^k} \leftarrow \mathsf{rArrayRead}(\{[\mathsf{attr}_i]_{2^k}\}_{i=0}^{m-1}, [\mathsf{idx}_j]_m)$ // 3 rounds & $4mk + 2m + 8k$ bits
9: (Comparison Phase)
10: $[\mathsf{v}_j < \mathsf{attr}_{\mathsf{idx}_j}]_2 \leftarrow \mathsf{NaiveRingLT}([\mathsf{v}_j]_{2^k}, [\mathsf{attr}_{\mathsf{idx}_j}]_{2^k})$ // $1 + \log_2(k)$ rounds & $3k + 3k \cdot \log_2(k)$ bits
11: $[\mathsf{v}_j == \mathsf{attr}_{\mathsf{idx}_j}]_2 \leftarrow \mathsf{NaiveRingEQ}([\mathsf{v}_j]_{2^k}, [\mathsf{attr}_{\mathsf{idx}_j}]_{2^k})$ // $1 + 2\log_2(k)$ rounds & $6k + 3k\log_2(k) - 3$ bits
12: $[\mathsf{comp}_j]_2 = [\mathsf{cond}_j]_2 \cdot [\mathsf{v}_j < \mathsf{attr}_{\mathsf{idx}_j}]_2 \oplus (1 \oplus [\mathsf{cond}_j]_2) \cdot [\mathsf{v}_j == \mathsf{attr}_{\mathsf{idx}_j}]_2$ // 1 round & 6 bits
13: **end for**
14: (Path Evaluation Phase)
15: Initialize $\mathsf{Path}_{j'} = \{[\mathsf{comp}_{\delta(j',0)}]_2, [\mathsf{comp}_{\delta(j',1)}]_2, \ldots, [\mathsf{comp}_{\delta(j',h-1)}]_2\}$ by picking up $[\mathsf{comp}_{\delta(j',\ell)}]_2$ corresponding to the intermediate value which has $(\ell+1)$-th height in the path to the j'-th leaf node from $\{[\mathsf{comp}_j]_2\}_{j=0}^{2^h-2}$.
16: **for** $j' = 0, \ldots, 2^h - 1$ **do in parallel**
17: $[\mathsf{pathBit}_{j'}]_2 \leftarrow \mathsf{rArrayAND}(\mathsf{Path}_{j'})$ // $\log_2(h)$ rounds & $3(h-1)$ bits
18: $[\mathsf{pathBit}_{j'}]_{2^k} \leftarrow \mathsf{BitConversion}([\mathsf{pathBit}_{j'}]_2)$ // 1 round & $6k$ bits
19: **end for**
20: return $[\mathsf{leafVal}]_{2^k} \leftarrow \mathsf{rInnerProduct}(\{[\mathsf{pathBit}_{j'}]_{2^k}\}_{j'=0}^{2^h-1}, \{[\mathsf{leafVal}_{j'}]_{2^k}\}_{j'=0}^{2^h-1})$. // 1 round & $3k$ bits

Protocol 2 Naive Construction of Private Decision Tree Evaluation via Secure Three-party Computation over Field

Input: Set of shared index number $\{[\![\mathsf{idx}_j]\!]_p\}_{j=0}^{2^h-2}$, shared feature array $\{[\![\mathsf{attr}_i]\!]_p\}_{i=0}^{m-1}$, thresh-

old values $\{[\![\mathsf{v}_j]\!]_p\}_{j=0}^{2^h-2}$, conditional values $\{[\![\mathsf{cond}_j]\!]_p\}_{j=0}^{2^h-2}$, set of shared values assigned to leaf nodes $\{[\![\mathsf{leafVal}_{j'}]\!]_p\}_{j'=0}^{2^h-1}$, mapping function δ

Output: shared values assigned to leaf node of correct path $[\![\mathsf{leafVal}_{j'}]\!]_p$ where j' is s.t. $\bigwedge_{\ell=0}^{h-1}(j'|_\ell \oplus \mathsf{comp}_{\delta(j',\ell)} \oplus 1) = 1$. Let $\mathsf{comp}_{\delta(j',\ell)}$ be $\mathsf{cond}_{\delta(j',\ell)} \cdot (\mathsf{v}_{\delta(j',\ell)} < \mathsf{attr}_{\mathsf{idx}_{\delta(j',\ell)}}) + (1 - \mathsf{cond}_{\delta(j',\ell)}) \cdot (\mathsf{v}_{\delta(j',\ell)} == \mathsf{attr}_{\mathsf{idx}_{\delta(j',\ell)}})$.

1: **for** $j = 0, \ldots, 2^h - 2$ **do**
2: (Feature Selection Phase)
3: $[\![\mathsf{attr}_{\mathsf{idx}_j}]\!]_p \leftarrow \mathsf{pArrayRead}(\{[\![\mathsf{attr}_i]\!]_p\}_{i=0}^{m-1}, [\![\mathsf{idx}_j]\!]_p)$ // 1 round & $(6\log_2(m) - 2) \cdot \log_2(p)$ bits in offline and 10 rounds & $(4m + 2\log_2(m) + 12) \cdot \log_2(p)$ bits in online
4: (Comparison Phase)
5: $[\![\mathsf{v}_j < \mathsf{attr}_{\mathsf{idx}_j}]\!]_p \leftarrow \mathsf{pLTZ}(k, [\![\mathsf{v}_j - \mathsf{attr}_{\mathsf{idx}_j}]\!]_p)$ // 1 round & $(3k-3) \cdot 6\log_2(p)$ bits in offline and 3 rounds & $(k+1) \cdot 6\log_2(p)$ bits in online
6: $[\![\mathsf{v}_j == \mathsf{attr}_{\mathsf{idx}_j}]\!]_p \leftarrow \mathsf{pEQZ}(k, [\![\mathsf{v}_j - \mathsf{attr}_{\mathsf{idx}_j}]\!]_p)$ // 1 round & $(k - 3\log_2(k) - 2) \cdot 6\log_2(p)$ bits in offline and 3 rounds & $(\log_2(k) + 2) \cdot 6\log_2(p)$ bits in online
7:

$$\begin{aligned} & [\![\mathsf{comp}_j]\!]_p \\ = \; & [\![\mathsf{cond}_j]\!]_p \cdot [\![\mathsf{v}_j < \mathsf{attr}_{\mathsf{idx}_j}]\!]_p \\ & +(1 - [\![\mathsf{cond}_j]\!]_p) \cdot [\![\mathsf{v}_j == \mathsf{attr}_{\mathsf{idx}_j}]\!]_p \\ = \; & [\![\mathsf{v}_j == \mathsf{attr}_{\mathsf{idx}_j}]\!]_p + [\![s_j]\!]_p \end{aligned}$$

where $[\![s_j]\!]_p \leftarrow \mathsf{pInnerProduct}(\{[\![\mathsf{cond}_j]\!]_p, [\![\mathsf{cond}_j]\!]_p\}, \{[\![\mathsf{v}_j < \mathsf{attr}_{\mathsf{idx}_j}]\!]_p, [\![\mathsf{v}_j == \mathsf{attr}_{\mathsf{idx}_j}]\!]_p\})$ // 1 round & $6\log_2(p)$ bits
8: **end for**
9: (Path Evaluation Phase)
10: Initialize $\mathsf{Path}_{j'} = \{[\![\mathsf{comp}_{\delta(j',0)}]\!]_p, [\![\mathsf{comp}_{\delta(j',1)}]\!]_p, \ldots, [\![\mathsf{comp}_{\delta(j',h-1)}]\!]_p\}$ by picking up $[\![\mathsf{comp}_{\delta(j',\ell)}]\!]_p$ correspond to the intermediate value which has $(\ell+1)$-th height in the path to the $j^{'}$-th leaf node from $\{[\![\mathsf{comp}_j]\!]_p\}_{j=0}^{2^h-2}$.
11: **for** $j^{'} = 0, \ldots, 2^h - 1$ **do**
12: **for** $\ell = 0, \ldots, h - 1$ **do**
13: $[\![c_{j^{'},\ell}]\!]_p = [\![j^{'}|_\ell \oplus \mathsf{comp}_{\delta(j',\ell)}]\!]_p = (j^{'}|_\ell - [\![\mathsf{comp}_{\delta(j',\ell)}]\!]_p)^2 = j^{'}|_\ell + [\![\mathsf{comp}_{\delta(j',0)}]\!]_p - 2 \cdot [\![\mathsf{comp}_{\delta(j',\ell)}]\!]_p$
14: **end for**
15: $[\![b_{j'}]\!]_p \leftarrow \mathsf{pArrayOr}(\{[\![c'_{j^{'},\ell}]\!]_p\}_{\ell=0}^{h-1})$ // 1 round & $\log_2(h) \cdot 6\log_2(p)$ bits in offline and 3 rounds & $3\log_2(h) \cdot 6\log_2(p)$ bits in online
16: $[\![\mathsf{pathBit}_{j'}]\!]_p = [\![1 \oplus b_{j'}]\!]_p = (1 - [\![b_{j'}]\!]_p)^2 = 1 + [\![b_{j'}]\!]_p - 2 \cdot [\![b_{j'}]\!]_p$
17: **end for**
18: return $[\![\mathsf{leafVal}]\!]_p \leftarrow \mathsf{pInnerProduct}(\{[\![\mathsf{pathBit}_{j'}]\!]_p\}_{j'=0}^{2^h-1}, \{[\![\mathsf{leafVal}_{j'}]\!]_p\}_{j'=0}^{2^h-1})$. // 1 round & $6\log_2(p)$ bits

4.2.13 Naive Construction of PDTE with Malicious Security and Fairness

To the best of our knowledge, Protocol 3 is the naive construction of the maliciously secure PDTE protocol with fairness based only on Trident [12], i.e., the best combination of the building blocks of Trident.

Protocol 3 Naive Construction of Private Decision Tree Evaluation via Trident

Input: Input attributes $\{[\mathsf{attr}_i]_{2^k}\}_{i=0}^{m-1}$, tree $\mathcal{T} = (h, \delta, \{[\mathsf{idx}_j]_{2^k}\}_{j=0}^{2^h-2}, \{[\mathsf{v}_j]_{2^k}\}_{j=0}^{2^h-2}, \{[\mathsf{cond}_j]_2\}_{j=0}^{2^h-2}, \{[\mathsf{leafVal}_{j'}]_{2^k}\}_{j'=0}^{2^h-1})$

Output: $[\mathsf{leafVal}_{j'}]_{2^k}$ where j' s.t. $\bigwedge_{\ell=0}^{h-1}(j'|_\ell == \mathsf{comp}_{\delta(j',\ell)}) = 1$. Let $\mathsf{comp}_{\delta(j',\ell)}$ be $\mathsf{cond}_{\delta(j',\ell)} \cdot (\mathsf{v}_{\delta(j',\ell)} < \mathsf{attr}_{\mathsf{idx}_{\delta(j',\ell)}}) \oplus (\mathsf{cond}_{\delta(j',\ell)} \oplus 1)(\mathsf{v}_{\delta(j',\ell)} == \mathsf{attr}_{\mathsf{idx}_{\delta(j',\ell)}})$.

1: **for** $j = 0, \ldots, 2^h - 2$ **do in parallel**
2: (Feature Selection Phase)
3: **for** $j'' = 0, \ldots, m-1$ **do in parallel**
4: P_3 sets $[0]_{2^k,3} = (0, 0, 0)$.
5: $P_{i'}$ $(i' = 0, 1, 2)$ sets $[0]_{2^k,i'} = (j'', 0, 0)$
6: $[\mathsf{idx}_j == j'']_2 \leftarrow \mathsf{NaiveEQ}([\mathsf{idx}_j]_{2^k}, [j'']_{2^k})$ // 2 rounds & $9k$ bits in offline, $2 + \log_2 \log_2 m$ rounds & $17 \log_2 m$ bits in online
7: $[\mathsf{idx}_j == j'']_{2^k} = \mathsf{BitConv}(2^k, [\mathsf{idx}_j == j'']_2)$ // 2 rounds & $3k$ bits in offline, 1 rounds & $3k + 1$ bits in online
8: **end for**
9: $[\mathsf{attr}_{\mathsf{idx}_j}]_{2^k} \leftarrow \mathsf{DotProd}(([\mathsf{attr}_0]_{2^k}, \ldots, [\mathsf{attr}_{m-1}]_{2^k}), ([\mathsf{idx}_j == 0]_{2^k}, \ldots, [\mathsf{idx}_j == m-1]_{2^k}))$ // 1 rounds & $3k$ bits in offline, 1 rounds & $3k$ bits in online
10: (Comparison Phase)
11: $[\mathsf{v}_j < \mathsf{attr}_{\mathsf{idx}_j}]_2 \leftarrow \mathsf{NaiveLT}([\mathsf{v}_j]_{2^k}, [\mathsf{attr}_{\mathsf{idx}_j}]_{2^k})$ // 1 round & $3k$ bits in offline, $1 + \log_2 k$ rounds & $7k$ bits in online
12: $[\mathsf{v}_j == \mathsf{attr}_{\mathsf{idx}_j}]_2 \leftarrow \mathsf{NaiveEQ}([\mathsf{v}_j]_{2^k}, [\mathsf{attr}_{\mathsf{idx}_j}]_{2^k})$ // 2 rounds & $9k$ bits in offline, $2 + \log_2 k$ rounds & $17k$ bits in online
13: $[\mathsf{comp}_j]_2 = [\mathsf{cond}_j]_2 \cdot [\mathsf{v}_j < \mathsf{attr}_{\mathsf{idx}_j}]_2 \oplus (1 \oplus [\mathsf{cond}_j]_2) \cdot [\mathsf{v}_j == \mathsf{attr}_{\mathsf{idx}_j}]_2 \leftarrow \mathsf{DotProd}(([\mathsf{cond}_j]_2, [1 \oplus \mathsf{cond}_j]_2), ([\mathsf{v}_j < \mathsf{attr}_{\mathsf{idx}_j}]_2, [\mathsf{v}_j == \mathsf{attr}_{\mathsf{idx}_j}]_2))$
// 1 round & 3 bits in offline, 1 round & 3 bits in online
14: **end for**
15: (Path Evaluation Phase)
16: Initialize $\mathsf{Path}_{j'} = \{[\mathsf{comp}_{\delta(j',0)}]_2, [\mathsf{comp}_{\delta(j',1)}]_2, \ldots, [\mathsf{comp}_{\delta(j',h-1)}]_2\}$ by picking up $[\mathsf{comp}_{\delta(j',\ell)}]_2$ correspond to the intermediate value which has $(\ell+1)$-th height in the path to the j'-th leaf node from $\{[\mathsf{comp}_j]_2\}_{j=0}^{2^h-2}$.
17: **for** $j' = 0, \ldots, 2^h - 1$ **do in parallel**
18: $[\mathsf{pathBit}_{j'}]_2 \leftarrow \mathsf{NaiveArrayAND}(\mathsf{Path}_{j'})$ // $\log_2 h$ rounds & $3(h-1)$ bits in offline, $\log_2 h$ rounds & $3(h-1)$ bits in online
19: $[\mathsf{pathBit}_{j'}]_{2^k} \leftarrow \mathsf{BitConv}(2^k, [\mathsf{pathBit}_{j'}]_2)$

// 2 rounds & $3k$ bits in offline, 1 rounds & $3k+1$ bits in online

20: **end for**

21: Return $[\mathsf{leafVal}]_{2^k} \leftarrow \mathsf{DotProd}(([\mathsf{pathBit}_0]_{2^k}, \ldots, [\mathsf{pathBit}_{2^h-1}]_{2^k}), ([\mathsf{leafVal}_0]_{2^k}, \ldots, [\mathsf{leafVal}_{2^h-1}]_{2^k}))$. // 1 rounds & $3k$ bits in offline, 1 rounds & $3k$ bits in online

4.2.14 Definition of Security for Fair SS-4PC

In order to define the security, we use the ideal/real simulation paradigm and define the real and ideal model in the almost same way as [8, 94] except that the number of parties. Note that we use the modified ideal model in [94] to guarantee fairness that the trusted party simply sends each party its output after the early abort option.

The real model: The parties execute a four-party protocol Π in the real model[12]. The adversary $\mathcal{A}$ (in a synchronous network) can be malicious and may deviate from the protocol specification, e.g., sending incorrect messages in place of the corrupted party and intentional delayed or unsent messages[13].

We let $\mathcal{A}$ be a non-uniform probabilistic polynomial-time adversary corrupting at most one party. That is, $\mathcal{A}$ can control $t < n/3$ party where $(t, n) = (1, 4)$ as a static corruption. We denote the set of parties as $\mathcal{P} = \{P_0, \ldots, P_3\}$. We also denote the output of the honest parties, the corrupted party $I \in \mathcal{P}$, and $\mathcal{A}$ in an real execution of Π, with inputs $x_0, \ldots, x_3$, auxiliary input z for $\mathcal{A}$, and security parameter κ by $\mathrm{REAL}_{\Pi,\mathcal{A}(z),I}(x_0, x_1, x_2, x_3, \kappa)$.

The ideal model: We define the ideal model, for any functionality $\mathcal{F}$, receiving inputs from $P_0, \ldots, P_3$ and providing them with outputs. The ideal execution proceeds as follows:

- **Send inputs to the trusted party:** Each honest party P_j $(j = 0, \ldots, 3)$ sends its specified input x_j to the trusted party. A corrupted party P_I controlled by $\mathcal{A}$ may either send its specified input x_I, some other x'_I or an abort message.
- **Early abort option:** If the trusted party received abort from $\mathcal{A}$, the trusted party sends $\perp$ to $\mathcal{P}$ and terminates. Otherwise, it proceeds to the next step.
- **Outputs:** The trusted party computes each party's output as specified by the functionality $\mathcal{F}$ based on the inputs received. We denote the output of P_j by y_j. Then, the trusted party sends y_j to P_j for $j = 0, \ldots, 3$. All parties (including the corrupted parties) always output the sent values from the trusted party that they obtained.

We define $\mathcal{S}$ as a non-uniform probabilistic polynomial-time adversary corrupting P_I. We denote the output of the honest parties, P_I, and $\mathcal{S}$ in an ideal execution with $\mathcal{F}$, with inputs $x_0, \ldots, x_4$, auxiliary input z for $\mathcal{S}$, and security parameter κ by $\mathrm{IDEAL}_{\mathcal{F},\mathcal{S}(z),I}(x_0, x_1, x_2, x_3, \kappa)$.

Definition 1. *Let $\mathcal{F}$ be a 4-party functionality, and let Π be a 4-party protocol. We say that Π computes $\mathcal{F}$ securely with fairness in the presence of one malicious party, if for*

[12] We assume that each party can be an input dealer and/or an output receiver. If a party distributes its input values as shares, he/she runs the input sharing protocol in Trident [12]. If a party reconstructs shares, he/she run the opening protocol, OpenOne.

[13] We assume that the parties know the maximum delay threshold for sending messages. Hence, honest parties can detect these cheating by waiting the maximum delay threshold for the data sent from the corrupted party and abort the protocol.

every non-uniform probabilistic polynomial-time adversary $\mathcal{A}$ in the real world, there exists a non-uniform probabilistic polynomial-time simulator $\mathcal{S}$ in the ideal model with $\mathcal{F}$, s.t. for every $i \in \{0,1,2,3\}$,

$$\begin{aligned}&\{\mathrm{IDEAL}_{\mathcal{F},\mathcal{S}(z),i}(x_0,x_1,x_2,x_3,\kappa)\}\\ \equiv&\{\mathrm{REAL}_{\Pi,\mathcal{A}(z),i}(x_0,x_1,x_2,x_3,\kappa)\}\end{aligned} \tag{4.1}$$

where $x_0,x_1,x_2,x_3 \in \{0,1\}^*$ under the constraint that $|x_0| = |x_1| = |x_2| = |x_3|$, $z \in \{0,1\}^*$ and $\kappa \in \mathbb{N}$.

Definition 1 is the modified security definition based on [8, 94]. If Eq. (4.1) holds with computational indistinguishability, then we say that Π computes $\mathcal{F}$ securely with computational security and fairness in the presence of one malicious party.

We also use the hybrid model to prove the security of the proposed schemes. In the hybrid model, each party runs the actual protocol with actual messages and can access the ideal subfunctionality computed by a trusted third party (TTP). As stated in [92], the subfunctionality computed by a TTP can be replaced with a secure real protocol. Let g be the subfunctionality. We say that protocol Π is secure in the g-hybrid model. We denote protocol Π that is secure in the g-hybrid model as Π^g.

4.3 Proposed Protocol with Semi-honest Security in Feature Selection Phase

4.3.1 Proposed Protocol with Semi-honest Security in Feature Selection Phase over Ring

In the feature selection phase, our protocol takes the share of the index $[\mathsf{idx}]_{2^k}$ (where $0 \leq \mathsf{idx} \leq m-1$) and the set of shared attributes $\{[\mathsf{attr}_j]_{2^k}\}_{j=0}^{m-1}$ as inputs and outputs $[\mathsf{attr}_{\mathsf{idx}}]_{2^k}$. To construct our feature selection protocol (Protocol 6), we propose a subprotocol, the bit-decomposition protocol (Protocol 5) that takes the bit length m and the shares $[x]_{2^k}$ as inputs and outputs the binary shares $\{[x|_j]\}_{j=0}^{m-1}$ of m bits. To construct the subprotocol, we also propose the MSB extraction protocol (Protocol 4). Therefore, we will explain, Protocosls 4, 5 and 6.

Protocol 4 $[\mathsf{msb}(a)]_2 \leftarrow \pi_{\mathsf{msbExt}}([a]_{2^k})$

Input: $[a]_{2^k}$ s.t. $a \in \mathbb{Z}_{2^k}, a = \sum_{j=0}^{k-1} 2^j \cdot a|_j$
Output: $[\mathsf{msb}(a)]_2 (= [a|_{k-1}]_2)$

1: (Offline phase)
2: **for** $j = 0, \ldots, k-1$ **do in parallel**
3: Each P_i runs $[r|_j]_{2,i} \leftarrow \mathsf{RndGen}(2, \mathsf{k}_i, \mathsf{k}_{i+1})$ where $r = \sum_{j=0}^{k-1} 2^j \cdot r|_j$. (for $i = 0,1,2$)
4: $[r|_j]_{2^k} \leftarrow \mathsf{BitConversion}([r|_j]_2)$ `// 1 round & 6k bits`
5: **end for**
6: $[r|_{k-2,\ldots,0}]_{2^k} = \sum_{j=0}^{k-2} 2^j \cdot [r|_j]_{2^k}$
7: $[2^{k-1} \cdot \mathsf{msb}(r)]_{2^k} = 2^{k-1} \cdot [r|_{k-1}]_{2^k}$
8: (Online phase)

9: $[a + (r|_{k-2,\ldots,0})]_{2^k} = [a]_{2^k} + [r|_{k-2,\ldots,0}]_{2^k}$
10: $[2 \cdot ((a+r)|_{k-2,\ldots,0})]_{2^k} = 2 \cdot [a + (r|_{k-2,\ldots,0})]_{2^k}$
11: **for** $i = 0, 1$ **do in parallel**
12: $\quad 2 \cdot ((a+r)|_{k-2,\ldots,0}) \leftarrow \mathsf{rOpen}(P_i,\ [2 \cdot ((a+r)|_{k-2,\ldots,0})]_{2^k})$ // 1 round & k bits
13: **end for**
14: $r|_{k-2,\ldots,0} \leftarrow \mathsf{rOpen}(P_2,\ [r|_{k-2,\ldots,0}]_{2^k})$ // 1 round & k bits
15: **for** $j = 0, \ldots, k-2$ **do in parallel**
16: $\quad \langle r|_j \rangle_{p',(0,1)} \leftarrow \mathsf{aShare}(P_2, p', r|_j, P_0, P_1)$ // 1 round & $\log_2(p')$ bits
17: **end for**
18: P_0 and P_1 generate $\beta \in \{0,1\}$, $s_j,\ u_j \in \mathbb{Z}_p^*$ by using F_p and k_1 for $j = 0, \ldots, k-2$.
19: $\beta \oplus (r|_{k-2,\ldots,0} > (a+r)|_{k-2,\ldots,0}) \leftarrow \mathsf{PrivateCompare}((a+r)|_{k-2,\ldots,0}, \{\langle r|_j \rangle_{p',(0,1)}\}_{j=0}^{k-2}, \beta, \{s_j\}_{j=0}^{k-2}, \{u_j\}_{j=0}^{k-2}, P_2)$ // 1 round & $2(k-1)\log_2(p')$ bits
20: P_0, P_1 and P_2 set $[\beta]_{2^k} = ((0,\beta),\ (\beta,0),\ (0,0))$
21: P_0, P_1 and P_2 set $[(a+r)|_{k-2,\ldots,0}]_{2^k} = ((0,(a+r)|_{k-2,\ldots,0}), ((a+r)|_{k-2,\ldots,0}, 0), (0,0))$.
22: $[r|_{k-2,\ldots,0}]_{2^k} \leftarrow \mathsf{rShare}(2^k, P_2, r|_{k-2,\ldots,0})$ // 1 round & $4k$ bits
23: $[u]_{2^k} = [\beta \oplus (r|_{k-2,\ldots,0} > (a+r)|_{k-2,\ldots,0})]_{2^k} \leftarrow \mathsf{rShare}(2^k, P_2, \beta \oplus (r|_{k-2,\ldots,0} > (a+r)|_{k-2,\ldots,0}))$ // 1 round and $4k$ bits
24: $[r|_{k-2,\ldots,0} > (a+r)|_{k-2,\ldots,0}]_{2^k} = ([u]_{2^k} - [\beta]_{2^k})^2$ // 1 round & $3k$ bits
25: $[a|_{k-2,\ldots,0}]_{2^k} = [(a+r)|_{k-2,\ldots,0}]_{2^k} - [r|_{k-2,\ldots,0}]_{2^k} + 2^{k-1} \cdot [r|_{k-2,\ldots,0} > (a+r)|_{k-2,\ldots,0}]_{2^k}$
26: $[2^{k-1} \cdot \mathsf{msb}(a)]_{2^k} = [2^{k-1} \cdot a|_{k-1}] = [a]_{2^k} - [a|_{k-2,\ldots,0}]_{2^k}$
27: $[2^{k-1} \cdot (\mathsf{msb}(a) \oplus \mathsf{msb}(r))]_{2^k} = [2^{k-1} \cdot \mathsf{msb}(a)]_{2^k} + [2^{k-1} \cdot \mathsf{msb}(r)]_{2^k} = 2^{k-1} \cdot [a|_{k-1}]_{2^k} + 2^{k-1} \cdot [r|_{k-1}]_{2^k}$
28: $2^{k-1} \cdot (\mathsf{msb}(a) \oplus \mathsf{msb}(r)) \leftarrow \mathsf{rOpen}(P_0, [2^{k-1} \cdot (\mathsf{msb}(a) \oplus \mathsf{msb}(r))]_{2^k})$ // 1 round & k bits
29: $[\mathsf{msb}(a) \oplus \mathsf{msb}(r)]_2 \leftarrow \mathsf{rShare}(2,\ P_0,\ \mathsf{msb}(a) \oplus \mathsf{msb}(r))$ // 1 round & 4 bits
30: $[\mathsf{msb}(a)]_2 = [\mathsf{msb}(a) \oplus \mathsf{msb}(r)]_2 \oplus [r|_{k-1}]_2$
31: return $[\mathsf{msb}(a)]_2$

Protocol 5 $\{[x|_j]_2\}_{j=0}^{m-1} \leftarrow \pi_{\mathsf{rBitDec}}(m, [x]_{2^k})$

Input: $m(\leq k)$, $[x]_{2^k}$ (s.t. $x \in \mathbb{Z}_{2^k},\ x = \sum_{j=0}^{k-1} 2^j \cdot x|_j,\ x|_j \in \mathbb{Z}_2$ for $j = 0, \ldots, k-1$).
Output: $\{[x|_j]_2\}_{j=0}^{m-1}$
1: **for** $j = 0, \ldots, m-1$ **do in parallel**
2: $\quad [2^{k-1-j} \cdot x|_{j,\ldots,0}]_{2^k} = 2^{k-1-j} \cdot [x]_{2^k}$
3: $\quad [x|_j]_2 = [\mathsf{msb}(2^{k-1-j} \cdot x|_{j,\ldots,0})]_2 \leftarrow \pi_{\mathsf{msbExt}}([2^{k-1-j} \cdot x|_{j,\ldots,0}]_{2^k})$ // 1 round & $6k^2$ bits in offline, 8 rounds & $9k + (3k-3)\log_2(p') + 4$ bits in online
4: **end for**
5: return $\{[x|_j]_2\}_{j=0}^{m-1}$

Protocol 6 $[\mathsf{attr}_{\mathsf{idx}}]_{2^k} \leftarrow \pi_{\mathsf{rFSelection}}([\mathsf{idx}]_{2^k},\ \{[\mathsf{attr}_j]_{2^k}\}_{j=0}^{m-1})$

Input: $[\mathsf{idx}]_{2^k},\ \{[\mathsf{attr}_j]_{2^k}\}_{j=0}^{m-1}$ (s.t. $0 \leq \mathsf{idx} < m < 2^k$).
Output: $[\mathsf{attr}_{\mathsf{idx}}]_{2^k}$
1: $\{[\mathsf{idx}|_i]_2\}_{i=0}^{\log_2(m)-1} \leftarrow \pi_{\mathsf{rBitDec}}(\log_2(m), [\mathsf{idx}]_{2^k})$ // 1 round & $6k^2 \log_2(m)$ bits in

offline, 8 rounds & $\log_2(m)(9k + (3k-3)\log_2(p') + 4)$ bits in online

2: **for** $i = 0, \ldots, \log_2(m) - 1$ **do in parallel**

3: $[\mathsf{idx}|_i]_m \leftarrow \mathsf{BitConversion}(m, [\mathsf{idx}|_i]_2)$ // 1 round & $6m$ bits

4: **end for**

5: return $[\mathsf{attr}_{\mathsf{idx}}]_{2^k} \leftarrow \mathsf{rArrayRead}(\{[\mathsf{attr}_i]_{2^k}\}_{i=0}^{m-1}, \sum_{i=0}^{\log_2(m)-1} 2^i \cdot [\mathsf{idx}|_i]_m)$ // 3 rounds & $4mk + 2m + 8k$ bits

---

Intuition behind Protocol 4. In the offline phase, each party prepares three types of the shares of random numbers $[r|_{k-2,\ldots,0}]_{2^k}, [2^{k-1} \cdot \mathsf{msb}(r)]_{2^k}, [r|_{k-1}]_2$ ($r = \sum_{j=0}^{k-1} 2^j \cdot r|_j, r|_j \in \mathbb{Z}_2$) (from Step 1 to 7).

In the online phase, our first goal is to compute $[a|_{k-2,\ldots,0}]_{2^k} (= [a \mod 2^{k-1}]_{2^k})$. First, each party uses $[r|_{k-2,\ldots,0}]_{2^k} (= [r \mod 2^{k-1}]_{2^k})$ to mask the input share $[a]_{2^k}$, then computes the masked share $[a + (r|_{k-2,\ldots,0})]_{2^k}$ (at Step 9). Next, each party computes $[2 \cdot ((a + r)|_{k-2,\ldots,0})]_{2^k} (= 2 \cdot [a + (r|_{k-2,\ldots,0})]_{2^k})$ (at Step 10). Then, P_0 and P_1 reveal $[2 \cdot ((a+r)|_{k-2,\ldots,0})]_{2^k}$ and obtain $2 \cdot ((a+r)|_{k-2,\ldots,0}) \mod 2^k$, from which they can obtain $(a+r)|_{k-2,\ldots,0}$ (from Step 11 to 13). We note that P_0 and P_1 cannot know the plain MSB bit, $\mathsf{msb}(a)$. In parallel, P_2 obtains $r|_{k-2,\ldots,0}$ (at Step 14). Next, each party runs PrivateCompare [33] and P_2 obtains the masked comarison result bit $\beta \oplus (r|_{k-2,\ldots,0} > (a + r)|_{k-2,\ldots,0})$ (from Step 15 to 19). Then, each party removes the mask β by performing arithmetic XOR operation over the ring (i.e., $([u]_{2^k} - [\beta]_{2^k})^2$ at Step 24), and obtains the share of comparison bit $[r|_{k-2,\ldots,0} > (a + r)|_{k-2,\ldots,0}]_{2^k}$ (from Step 20 to 24). Finally, each party computes $[a|_{k-2,\ldots,0}]_{2^k} = [(a + r)|_{k-2,\ldots,0}]_{2^k} - [r|_{k-2,\ldots,0}]_{2^k} + 2^{k-1} \cdot [r|_{k-2,\ldots,0} > (a + r)|_{k-2,\ldots,0}]_{2^k} (= [(a+r) \mod 2^{k-1}]_{2^k} - [r \mod 2^{k-1}]_{2^k} + 2^{k-1} \cdot [(r \mod 2^{k-1}) > ((a+r) \mod 2^{k-1})]_{2^k})$.

We note that $[a|_{k-2,\ldots,0}]_{2^k} = [(a + r)|_{k-2,\ldots,0}]_{2^k} - [r|_{k-2,\ldots,0}]_{2^k}$ does not hold in general. The reason that the equation does not work is because the wrap-around phenomenon may occur. The wrap-around phenomenon means that the modulo operation may make $(a + r)|_{k-2,\ldots,0} \mod 2^k$ less than $r|_{k-2,\ldots,0} \mod 2^k$. To deal with a case where wrap-around phenomenon occurs, we must compute $[(a+r)|_{k-2,\ldots,0}]_{2^k} - [r|_{k-2,\ldots,0}]_{2^k} + 2^{k-1} \cdot [r|_{k-2,\ldots,0} > (a + r)|_{k-2,\ldots,0}]_{2^k}$ similarly to [81–83] to ignore the effect of the wrap-around phenomenon.

Next, each party computes $[2^{k-1} \cdot \mathsf{msb}(a)]_{2^k} = [2^{k-1} \cdot a|_{k-1}]_{2^k} = [a]_{2^k} - [a|_{k-2,\ldots,0}]_{2^k} = [a]_{2^k} - [a \mod 2^{k-1}]_{2^k}$ (at Step 26). Then, P_0 obtains $2^{k-1} \cdot (\mathsf{msb}(a) \oplus \mathsf{msb}(r))$, from which P_0 can obtain $\mathsf{msb}(a) \oplus \mathsf{msb}(r)$ (from Step 27 and 28). P_0 distributes $[\mathsf{msb}(a) \oplus \mathsf{msb}(r)]_2$ (at Step 29). Finally, each party computes $[\mathsf{msb}(a)]_2 = [\mathsf{msb}(a) \oplus \mathsf{msb}(r)]_2 \oplus [r|_{k-1}]_2$.

Intuitions behind Protocols 5 and 6. In Protocol 5, each party computes the left shift (at Step 2) and invokes π_{msbExt} (at Step 3), repeatedly. Then, each party gets the shares of each bit over $\mathbb{Z}_2$.

In Protocol 6, to convert $[\mathsf{idx}]_{2^k}$ to $[\mathsf{idx}]_m$, each party invokes π_{rBitDec} and runs BitConversion (from Step 1 to Step 4). Then, each party uses the OAR protocol [60], rArrayRead and obtains $[\mathsf{attr}_{\mathsf{idx}}]_{2^k}$ (at Step 5).

We emphasize that it is non-trivial to convert $[\mathsf{idx}]_{2^k}$ to $[\mathsf{idx}]_m$ with constant rounds. For example, if we use the (naive) circuit-based bit-decomposition protocol, BitDec, and BitConversion, the conversion of the shared index is not a constant-round protocol. Hence, the whole of the feature selection protocol is also not a constant-round protocol if the conversion of the shared index is not a constant-round protocol.

4.3.2 Proposed Protocol with Semi-honest Security in Feature Selection Phase over Field

Protocol 7 $[\![\mathsf{attr}_{\mathsf{idx}}]\!]_p \leftarrow \pi_{\mathsf{pFSelection}}([\![\mathsf{idx}]\!]_p,\ \{[\![\mathsf{attr}_j]\!]_p\}_{j=0}^{m-1})$

Input: $[\![\mathsf{idx}]\!]_p,\ \{[\![\mathsf{attr}_j]\!]_p\}_{j=0}^{m-1}$ (s.t. $0 \leq \mathsf{idx} < m \leq 2^{k-1} \leq p$).
Output: $[\![\mathsf{attr}_{\mathsf{idx}}]\!]_p$

1: **for** $j = 0, \ldots, m-1$ **do**
2: (Offline Phase)
3: P_i chooses $r_{j,i} \in \mathbb{F}_p^*$ randomly. (for $i = 0, 1, 2$)
4: $[\![r_{j,i}]\!]_p \leftarrow \mathsf{pShare}(P_i, r_{j,i})$ (for $i = 0, 1, 2$) // 1 round and $6\log_2(p)$ bits
5: $[\![r_j]\!]_p = [\![r_{j,0}]\!]_p \cdot [\![r_{j,1}]\!]_p \cdot [\![r_{j,2}]\!]_p$ // 2 rounds and $12\log_2(p)$ bits
6: (Online Phase)
7: $[\![d_j]\!]_p = [\![r_j \cdot (\mathsf{idx} - j)]\!]_p = [\![r_j]\!]_p \cdot ([\![\mathsf{idx}]\!]_p - j)$ // 1 round and $6\log_2(p)$ bits
8: **end for**
9: Initialize $\mathcal{R}_j = \{[\![\mathsf{attr}_j]\!]_p, [\![d_j]\!]_p\}$ for $j = 0, \ldots, m-1$.
10: $\{\mathcal{R}'_j\}_{j=0}^{m-1} \leftarrow \mathsf{pSetShuffle}(\{\mathcal{R}_j\}_{j=0}^{m-1})$ // 6 round and $18m\log_2(p)$ bits
11: **for** $j = 0, \ldots, m-1$ **do**
12: Pick up $[\![d'_j]\!]_p$ from $\mathcal{R}'_j = \{[\![\mathsf{attr}'_j]\!]_p, [\![d'_j]\!]_p\}$.
13: $d'_j \leftarrow \mathsf{pOpen}(P_i, [\![d'_j]\!]_p)$ (for $i = 0, 1, 2$) // 1 round and $3\log_2(p)$ bits
14: **end for**
15: return $[\![\mathsf{attr}'_j]\!]_p$ where $d'_j = 0$.

Intuition behind Protocol 7. In offline phase, each party generates the shares of non-zero random values $\{[\![r_j]\!]_p\}_{j=0}^{m-1}$ (from Step 1 to Step5). Next, let d_j be the value to indicate whether j and idx are matched. If it holds that $j = \mathsf{idx}$, d_j equals 0. If it holds that $j \neq \mathsf{idx}$, d_j does not equal 0. Each party computes the share of d_j, $[\![d_j]\!]_p$ (at Step 7).

Next, each party initializes $\mathcal{R}_j = \{[\![\mathsf{attr}_j]\!]_p, [\![d_j]\!]_p\}$ for each j and computes the obliviously shuffled set of $\mathcal{R}_j$, $\{\mathcal{R}'_j\}_{j=0}^{m-1}$ where $\mathcal{R}'_j = \{[\![\mathsf{attr}'_j]\!]_p, [\![d'_j]\!]_p\}$ such that $\mathsf{attr}'_{\sigma(j)} = \mathsf{attr}_j$, $d'_{\sigma(j)} = d_j$ and $\mathcal{R}'_{\sigma(j)} = \mathcal{R}_j$ while no one knows the random permutation σ (at Steps 9 and 10). Then, each party reveals d'_j (at Steps 12 and 13). If it holds that $j = \mathsf{idx}$, $d'_{\sigma(j)}$ equals to 0. If it holds that $j \neq \mathsf{idx}$, $d'_{\sigma(j)}$ does not equal to 0 (i.e., non-zero random value). We note that the revealed value d'_j does not leak the positional information and the information about idx because d'_j is the obliviously shuffled value by the random permutation σ that is known to no one. Finally, each party outputs $[\![\mathsf{attr}'_j]\!]_p$ (where $d'_j = 0$).

4.4 Proposed Protocol with Semi-honest Security in Comparison Phase

Protocol 8 $[a < b]_2 \leftarrow \pi_{\mathsf{rLT}}([a]_{2^k},\ [b]_{2^k})$

Input: $[a]_{2^k},\ [b]_{2^k}$ (s.t. $0 \leq a, b \leq 2^{k-1} - 1$)
Output: $[a < b]_2$

1: $[c]_{2^k} = [a]_{2^k} - [b]_{2^k}$

2: $[\mathsf{msb}(c)]_2 \leftarrow \pi_{\mathsf{msbExt}}([c]_{2^k})$ // 1 round & $6k^2$ bits in offline, 8 rounds & $9k+(3k-3)\log_2(p')+4$ bits in online

3: return $[\mathsf{msb}(c)]_2$

Protocol 9 $[a == b]_2 \leftarrow \pi_{\mathsf{rEQ}}([a]_{2^k},\ [b]_{2^k})$

Input: $[a]_{2^k},\ [b]_{2^k}$ (s.t. $0 \le a, b \le 2^{k-1}-1$)
Output: $[a == b]_2$

1: $[a < b]_2 \leftarrow \pi_{\mathsf{rLT}}([a]_{2^k},\ [b]_{2^k})$ // 1 round & $6k^2$ bits in offline, 8 rounds & $9k+(3k-3)\log_2(p')+4$ bits in online

2: $[b < a]_2 \leftarrow \pi_{\mathsf{rLT}}([b]_{2^k},\ [a]_{2^k})$ // 1 round & $6k^2$ bits in offline, 8 rounds & $9k+(3k-3)\log_2(p')+4$ bits in online

3: $[\mathsf{res}]_2 = ([a < b]_2 \oplus 1) \cdot ([b < a]_2 \oplus 1)$ // 1 round & 3 bits

4: return $[\mathsf{res}]_2$

Protocol 10 $\{[\mathsf{comp}_j]_2\}_{j=0}^{2^h-2} \leftarrow \pi_{\mathsf{rComp}}(\{[\mathsf{attr}_{\mathsf{idx}_j}]_{2^k}\}_{j=0}^{2^h-2}, \{[\mathsf{v}_j]_{2^k}\}_{j=0}^{2^h-2}, \{[\mathsf{cond}_j]_2\}_{j=0}^{2^h-2})$

Input: Attribute set $\{[\mathsf{attr}_{\mathsf{idx}_j}]_{2^k}\}_{j=0}^{2^h-2}$, threshold values $\{[\mathsf{v}_j]_{2^k}\}_{j=0}^{2^h-2}$, conditional values $[\mathsf{cond}_j]_2\}_{j=0}^{2^h-2}$
Output: $\{[\mathsf{comp}_j]_2\}_{j=0}^{2^h-2}$

1: **for** $j = 0, \ldots, 2^h-2$ **do in parallel**

2: $[\mathsf{v}_j < \mathsf{attr}_{\mathsf{idx}_j}]_2 \leftarrow \pi_{\mathsf{rLT}}([\mathsf{v}_j]_{2^k}, [\mathsf{attr}_{\mathsf{idx}_j}]_{2^k})$ // 1 round & $6k^2$ bits in offline, 8 rounds & $9k+(3k-3)\log_2(p')+4$ bits in online

3: $[\mathsf{v}_j == \mathsf{attr}_{\mathsf{idx}_j}]_2 \leftarrow \pi_{\mathsf{rEQ}}([\mathsf{v}_j]_{2^k}, [\mathsf{attr}_{\mathsf{idx}_j}]_{2^k})$ // 1 round & $12k^2$ bits in offline, 9 rounds & $18k+(6k-6)\log_2(p')+11$ bits in online

4: $[\mathsf{comp}_j]_2 = [\mathsf{cond}_j]_2 \cdot [\mathsf{v}_j < \mathsf{attr}_{\mathsf{idx}_j}]_2 \oplus (1 \oplus [\mathsf{cond}_j]_2) \cdot [\mathsf{v}_j == \mathsf{attr}_{\mathsf{idx}_j}]_2$ // 1 round & 6 bits

5: **end for**

6: return $\{[\mathsf{comp}_j]_2\}_{j=0}^{2^h-2}$

In the comparison phase, it is required to compare the attribute value and the decision threshold value without revealing these values, the comparison operators, and comparison results. Hence, we construct the less-than protocol (Protocol 8) and the equality-testing protocol (Protocol 9) as subprotocols. Then, we also construct the comparison protocol (Protocol 10) by using these subprotocols, i.e., oblivious selection protocol for the comparison result.

In Protocol 8, to run the less-than protocol, each party computes $[a-b]_{2^k} = [a]_{2^k} - [b]_{2^k}$ where we assume $0 \le a, b \le 2^{k-1}-1$ and invokes π_{msbExt}. If a is smaller than b, $\mathsf{msb}(a-b)$ equals 1 and can be outputted as the result of the less-than protocol. If not, $\mathsf{msb}(a-b)$ equals 0 and can be outputted. In Protocol 9, to run the equality-testing protocol, each party invokes $\pi_{\mathsf{rLT}}([a]_{2^k}, [b]_{2^k})$ and $\pi_{\mathsf{rLT}}([b]_{2^k}, [a]_{2^k})$ in parallel. $a = b$ holds if $(a < b) \oplus 1 = 1$ and $(b < a) \oplus 1 = 1$. Therefore, each party outputs $([a < b]_2 \oplus 1) \cdot ([b < a]_2 \oplus 1)$ as the

result of the equality-testing protocol.

Protocol 10 takes the shares of the conditional values $\{[\mathsf{cond}_j]_2\}_{j=0}^{2^h-2}$ as inputs and outputs the oblivious comparison results for each internal node. It either outputs the results of π_{rLT} (if $c_j = 1$) or π_{rEQ} (if $c_j = 0$). Hence, each party invokes π_{rLT} and π_{rEQ} (at Steps 2 and 3). Then, each party selects one result or the other as the shared comparison result bit $[\mathsf{comp}_j]_2$, obliviously (at Step 4).

4.5 Proposed Protocol with Semi-honest Security in Path Evaluation Phase

4.5.1 Proposed Path Evaluation Protocol with Semi-honest Security over Ring

Protocol 11 $[\mathsf{leafVal}_{j'}]_{2^k} \leftarrow \pi_{\mathsf{rPathEval}}(\{[\mathsf{comp}_j]_2\}_{j=0}^{2^h-2}, \{[\mathsf{leafVal}_{j'}]_{2^k}\}_{j'=0}^{2^h-1}, \delta)$

Input: Comparison result of intermediate nodes $\{[\mathsf{comp}_j]_2\}_{j=0}^{2^h-2}$, set of shared values assigned to leaf nodes $\{[\mathsf{leafVal}_{j'}]_{2^k}\}_{j'=0}^{2^h-1}$, mapping function δ

Output: shared values assigned to leaf node of correct path $[\mathsf{leafVal}_{j'}]_{2^k}$ where j' is s.t. $\bigwedge_{\ell=0}^{h-1}(j'|_\ell == \mathsf{comp}_{\delta(j',\ell)}) = 1$.

1: **for** $j' = 0, \ldots, 2^h - 1$ **do in parallel**
2: Initialize $\mathsf{Path}_{j'} = \{[\mathsf{comp}_{\delta(j',0)}]_2, [\mathsf{comp}_{\delta(j',1)}]_2, \ldots, [\mathsf{comp}_{\delta(j',h-1)}]_2\}$.
3: **for** $\ell = 0, \ldots, h-1$ **do in parallel**
4: Compute $[c_{j',\ell}]_2 = [j'|_\ell == \mathsf{comp}_{\delta(j',\ell)}]_2 = j'|_\ell \oplus [\mathsf{comp}_{\delta(j',\ell)}]_2 \oplus 1$ by picking up $[\mathsf{comp}_{\delta(j',\ell)}]_2$ from $\mathsf{Path}_{j'}$.
5: **end for**
6: Set $\mathcal{R}_{j'} = \{[\mathsf{leafVal}_{j'}]_{2^k}, [c_{j',0}]_2, \ldots, [c_{j',h-1}]_2\}$
7: **end for**
8: $\{\mathcal{R}'_{j'}\}_{j'=0}^{2^h-1} \leftarrow \mathsf{rSetShuffle}(\{\mathcal{R}_{j'}\}_{j'=0}^{2^h-1})$ // 3 rounds & $6 \cdot 2^h \cdot (k+h)$ bits
9: Initialize $\mathsf{count}_{j'} = 0$ for $j' = 0, \ldots, 2^h - 1$.
10: **for** $j' = 0, \ldots, 2^h - 1; \ell = 0, \ldots, h-1$ **do in parallel**
11: Pick up $[c'_{j',\ell}]_2$ from $\mathcal{R}'_{j'} = \{[\mathsf{leafVal}'_{j'}]_{2^k}, [c'_{j',0}]_2, \ldots, [c'_{j',h-1}]_2\}$. Then, $c'_{j',\ell} \leftarrow \mathsf{Open}(P_i, [c'_{j',\ell}]_2)$ for $i = 0, 1, 2$. // 1 round & 3 bits
12: $\mathsf{count}_{j'} = \mathsf{count}_{j'} + 1$ if $c_{j',\ell} = 1$.
13: **end for**
14: return $[\mathsf{leafVal}'_{j'}]_{2^k}$ where $\mathsf{count}'_{j'} = h$.

In path evaluation phase, it is required to choose the correct path (and leaf node) by using the shared comparison result of each node and the shared class labels assigned to leaf nodes without revealing the information about paths, comparison results, and class labels. We construct the path evaluation protocol (Protocol 11) and explain the intuition behind it. Each party initializes $\mathsf{Path}_{j'} = \{[\mathsf{comp}_{\delta(j',\ell)}]_2\}_{\ell=0}^{h-1}$ for each j'-th leaf node (at Step 2). δ is the mapping function defined in §4.2.10. Next, let $[\mathsf{leafVal}_{j'}]_{2^k}$ be the shares of each class

label value and $c_{j',\ell}$ be the result of the equality-testing between the bit assigned to the branch that has the ℓ-th height on the path to the j'-th leaf node, $j'|_\ell$ and the comparison result $\mathsf{comp}_{\delta(j',\ell)}$. Each party computes $\mathcal{R}_{j'} = \{[\mathsf{leafVal}_{j'}]_{2^k}, [c_{j',0}]_2, \ldots, [c_{j',h-1}]_2\}$ (from Step 1 to 6).

Next, each party computes $\{\mathcal{R}'_{j'}\}_{j'=0}^{2^h-1}$ by shuffling $\{\mathcal{R}_{j'}\}_{j'=0}^{2^h-1}$ obliviously by using the random permutation σ such that $\mathcal{R}'_{\sigma(j')} = \mathcal{R}_{j'}$ (at Step 8). After initializing $\mathsf{count}_{j'} = 0$ (at Step 9). Then, each party obtains $c'_{j',\ell}$ by choosing $[c'_{j',\ell}]_2$ from $\mathcal{R}'_{j'}$ and revealing it. If $c'_{j',\ell} = 1$, each party increases $\mathsf{count}_{j'}$ by 1 (from Step 10 to Step 13). We note that $c'_{j',\ell}$ does not leak the positional information j'. In addition, an adversary can obtain no information about $\{\mathsf{comp}_{\delta(j',\ell)}\}_{\ell=0}^{2^h-2}$ and $\mathsf{leafVal}_{j'}$ from $c'_{j',\ell}$ because the tree is a complete binary tree. For example, we assume that $h = 2$. If the correct output leaf node is the leaf node $2(= 10_{(2)})$, it holds that $c'_{0,0} = 1$, $c'_{0,1} = 0$, $c'_{1,0} = 0$, $c'_{1,1} = 0$, $c'_{2,0} = 1$, $c'_{2,1} = 1$, $c'_{3,0} = 0$, and $c'_{3,1} = 1$. That is, the adversary gets all the 2-bit sequences, $00_{(2)}$, $01_{(2)}$, $10_{(2)}$, and $11_{(2)}$. As another example, if the correct output leaf node is the leaf node $3(= 11_{(2)})$, it holds that $c'_{0,0} = 0$, $c'_{0,1} = 0$, $c'_{1,0} = 1$, $c'_{1,1} = 0$, $c'_{2,0} = 0$, $c'_{2,1} = 1$, $c'_{3,0} = 1$, and $c'_{3,1} = 1$. In this case, the adversary also obtains all 2-bit sequences, $00_{(2)}$, $01_{(2)}$, $10_{(2)}$, and $11_{(2)}$. We note that the random permutation σ is different for each execution of $\mathsf{rSetShuffle}$. Therefore, the adversary can get no information from $c'_{j',\ell}$.

If $\mathsf{count}_{\sigma(j')} = h$, $\bigwedge_{\ell=0}^{h-1}(j'|_\ell == \mathsf{comp}_{\delta(j',\ell)}) = \bigwedge_{\ell=0}^{h-1}(j'|_\ell \oplus \mathsf{comp}_{\delta(j',\ell)} \oplus 1) = 1$ holds. Therefore, each party outputs $[\mathsf{leafVal}'_{j'}]_{2^k}$ where $\mathsf{count}_{j'} = h$ (at Step 14).

4.5.2 Proposed Path Evaluation Protocol with Semi-honest Security over Field

Protocol 12 $[\![\mathsf{leafVal}]\!]_p \leftarrow \pi_{\mathsf{pPathEval}}(\{[\![\mathsf{comp}_j]\!]_p\}_{j=0}^{2^h-2}, \{[\![\mathsf{leafVal}_{j'}]\!]_p\}_{j'=0}^{2^h-1}, \delta)$

Input: Comparison result of intermediate nodes $\{[\![\mathsf{comp}_j]\!]_p\}_{j=0}^{2^h-2}$, set of shared values assigned to leaf nodes $\{[\![\mathsf{leafVal}_{j'}]\!]_p\}_{j'=0}^{2^h-1}$, mapping function δ

Output: shared values assigned to leaf node of correct path $[\![\mathsf{leafVal}_{j'}]\!]_p$ where j' is s.t. $\bigwedge_{\ell=0}^{h-1}(j'|_\ell == \mathsf{comp}_{\delta(j',\ell)}) = 1$.

1: (Offline phase)
2: **for** $j' = 0, \ldots, 2^h - 1$ **do**
3: P_i chooses $r_{j',i} \in \mathbb{F}_p^*$ randomly. (for $i = 0, 1, 2$)
4: $[\![r_{j',i}]\!]_p \leftarrow \mathsf{pShare}(P_i, r_{j',i})$ (for $i = 0, 1, 2$) // 1 round and $6\log_2(p)$ bits
5: $[\![r_{j'}]\!]_p = [\![r_{j',0}]\!]_p \cdot [\![r_{j',1}]\!]_p \cdot [\![r_{j',2}]\!]_p$ // 2 rounds and $12\log_2(p)$ bits
6: **end for**
7: (Online phase)
8: Initialize $\mathsf{Path}_{j'} = \{[\![\mathsf{comp}_{\delta(j',0)}]\!]_p, [\![\mathsf{comp}_{\delta(j',1)}]\!]_p, \ldots, [\![\mathsf{comp}_{\delta(j',h-1)}]\!]_p\}$ by picking up $[\![\mathsf{comp}_{\delta(j',\ell)}]\!]_p$ corresponding to the intermediate value which has $(\ell+1)$-th height in the path to the j'-th leaf node from $\{[\![\mathsf{comp}_j]\!]_p\}_{j=0}^{2^h-2}$.
9: **for** $j' = 0, \ldots, 2^h - 1$ **do**

10: **for** $\ell = 0, \ldots, h-1$ **do**
11: Compute $[\![c_{j',\ell}]\!]_p = [\![j'|_\ell \oplus \mathsf{comp}_{\delta(j',\ell)}]\!]_p = (j'|_\ell - [\![\mathsf{comp}_{\delta(j',\ell)}]\!]_p)^2 = j'|_\ell + [\![\mathsf{comp}_{\delta(j',\ell)}]\!]_p - 2 \cdot [\![\mathsf{comp}_{\delta(j',\ell)}]\!]_p$ by picking up $[\![\mathsf{comp}_{\delta(j',\ell)}]\!]_p$ from $\mathsf{Path}_{j'}$.
12: **end for**
13: $[\![\mathsf{sum}_{j'}]\!]_p = \sum_{\ell=0}^{h-1} [\![c_{j',\ell}]\!]_p$
14: $[\![r_{j'} \cdot \mathsf{sum}_{j'}]\!]_p = [\![r_{j'}]\!]_p \cdot [\![\mathsf{sum}_{j'}]\!]_p$ // 1 round and 6log(p) bits
15: Initialize $\mathcal{R}_{j'} = \{[\![\mathsf{leafVal}_{j'}]\!]_p, [\![r_{j'} \cdot \mathsf{sum}_{j'}]\!]_p\}$
16: **end for**
17: $\{\mathcal{R}'_{j'}\}_{j'=0}^{2^h-1} \leftarrow \mathsf{pSetShuffle}(\{\mathcal{R}_{j'}\}_{j'=0}^{2^h-1})$ // 6 rounds and $18 \cdot 2^h \cdot \log_2(p)$ bits
18: **for** $j' = 0, \ldots, 2^h - 1$ **do**
19: Pick up $[\![r'_{j'} \cdot \mathsf{sum}'_{j'}]\!]_p$ from $\mathcal{R}'_{j'} = \{[\![\mathsf{leafVal}'_{j'}]\!]_p, [\![r'_{j'} \cdot \mathsf{sum}'_{j'}]\!]_p\}$.
20: $r'_{j'} \cdot \mathsf{sum}'_{j'} \leftarrow \mathsf{pOpen}(P_i, [\![r'_{j'} \cdot \mathsf{sum}'_{j'}]\!]_p)$ (for $i = 0, 1, 2$) // 1 round and $2^h \cdot 3\log_2(p)$ bits
21: **end for**
22: return $[\![\mathsf{leafVal}'_{j'}]\!]_p$ where $r'_{j'} \cdot \mathsf{sum}'_{j'} = 0$.

In offline phase, each party computes the shares of non-zero random values $\{[\![r_{j'}]\!]_p\}_{j'=0}^{2^h-1}$ (from Step 1 to Step 6). Next, let $[\![\mathsf{leafVal}_{j'}]\!]_p$ be the shares of the class label value assigned to the j'-th leaf node. Let also $c_{j',\ell}$ be the equality-testing result between the bit assigned to the branch that has the ℓ-th height on the path to the j'-th leaf node, $j'|_\ell$ and the comparison result $\mathsf{comp}_{\delta(j',\ell)}$. We set $\mathsf{sum}_{j'} = \sum_{\ell=0}^{h-1} c_{j',\ell}$. Each party computes $\mathcal{R}_{j'} = \{[\![\mathsf{leafVal}_{j'}]\!]_p, [\![r_{j'} \cdot \mathsf{sum}_{j'}]\!]_p\}$ (from Step 7 to Step 16).

Next, each party computes $\{\mathcal{R}'_{j'}\}_{j'=0}^{2^h-1}$ by shuffling $\{\mathcal{R}_{j'}\}_{j'=0}^{2^h-1}$ obliviously by using the random permutation σ such that $\mathcal{R}'_{\sigma(j')} = \mathcal{R}_{j'}$, $[\![\mathsf{leafVal}'_{\sigma(j')}]\!]_p = [\![\mathsf{leafVal}_{j'}]\!]_p$, and $[\![r'_{\sigma(j')} \cdot \mathsf{sum}'_{\sigma(j')}] = [\![r_{j'} \cdot \mathsf{sum}_{j'}]$ (at Step 17). each party gets $r'_{j'} \cdot \mathsf{sum}'_{j'}$ by choosing its shares from $\{\mathcal{R}'_{j'}\}_{j'=0}^{2^h-1}$ and revealing it (from Step 18 to Step 21). If $\bigwedge_{\ell=0}^{h-1}(j'|_\ell == \mathsf{comp}_{\delta(j',\ell)}) = 1$, i.e., $\bigvee_{\ell=0}^{h-1} c_{j',\ell} = 0$, then $\mathsf{sum}'_{\sigma(j')} = 0$. If not, then $\mathsf{sum}'_{\sigma(j')} \neq 0$. That is, it holds that $r'_{\sigma(j')} \cdot \mathsf{sum}'_{\sigma(j')} = 0$ iff $\bigwedge_{\ell=0}^{h-1}(j'|_\ell == \mathsf{comp}_{\delta(j',\ell)}) = 1$. Therefore, it outputs $[\![\mathsf{leafVal}'_{\sigma(j')}]\!]_p$ where $r'_{\sigma(j')} \cdot \mathsf{sum}'_{\sigma(j')} = 0$ (at Step 22). We note that $r'_{\sigma(j')} \cdot \mathsf{sum}'_{\sigma(j')}$ leaks no positional information about j' because of $\mathsf{pSetShuffle}$.

4.6 Proposed Protocol of Private Decision Tree Evaluation with Semi-honest Security

4.6.1 Proposed Protocol of Private Decision Tree Evaluation with Semi-honest Security over Ring

Protocol 13 $[\mathsf{leafVal}]_{2^k} \leftarrow \pi_{\mathsf{rDTEval}}(\{[\mathsf{idx}_j]_{2^k}\}_{j=0}^{2^h-2}, \{[\mathsf{attr}_i]_{2^k}\}_{i=0}^{m-1}, \{[\mathsf{v}_j]_{2^k}\}_{j=0}^{2^h-2}, \{[\mathsf{cond}_j]_{2^k}\}_{j=0}^{2^h-2}, \{[\mathsf{leafVal}_{j'}]_{2^k}\}_{j'=0}^{2^h-1}, \delta)$

Input: Set of shared index number $\{[\mathsf{idx}_j]_{2^k}\}_{j=0}^{2^h-2}$, shared feature array $\{[\mathsf{attr}_i]_{2^k}\}_{i=0}^{m-1}$, threshold values $\{[\mathsf{v}_j]_{2^k}\}_{j=0}^{2^h-2}$, conditional values $\{[\mathsf{cond}_j]_{2^k}\}_{j=0}^{2^h-2}$, set of shared values assigned to leaf nodes $\{[\mathsf{leafVal}_{j'}]_{2^k}\}_{j'=0}^{2^h-1}$, mapping function δ

Output: shared values assigned to leaf node of correct path $[\mathsf{leafVal}_{j'}]_{2^k}$ where j' is s.t. $\bigwedge_{\ell=0}^{h-1}(j'|_\ell \oplus \mathsf{comp}_{\delta(j',\ell)} \oplus 1) = 1$. Let $\mathsf{comp}_{\delta(j',\ell)}$ be $\mathsf{cond}_{\delta(j',\ell)} \cdot (\mathsf{v}_{\delta(j',\ell)} < \mathsf{attr}_{\mathsf{idx}_{\delta(j',\ell)}}) + (1 - \mathsf{cond}_{\delta(j',\ell)}) \cdot (\mathsf{v}_{\delta(j',\ell)} == \mathsf{attr}_{\mathsf{idx}_{\delta(j',\ell)}})$.

1: Initialize $\mathcal{A}$.
2: **for** $j = 0, \ldots, 2^h - 2$ **do in parallel**
3: $[\mathsf{attr}_{\mathsf{idx}_j}]_{2^k} \leftarrow \pi_{\mathsf{rFSelection}}([\mathsf{idx}_j]_{2^k}, \{[\mathsf{attr}_i]_{2^k}\}_{i=0}^{m-1})$. // 1 round & $6k^2\log_2(m)$ bits in offline, 12 rounds & $\log_2(m)(9k + (3k-3)\log_2(p') + 4) + 6m\log_2(m) + 4mk + 2m + 8k$ bits in online
4: Set $[\mathsf{attr}_{\mathsf{idx}_j}]_{2^k}$ into $\mathcal{A}$.
5: **end for**
6: $\{[\mathsf{comp}_j]_2\}_{j=0}^{2^h-2} \leftarrow \pi_{\mathsf{rComp}}(\mathcal{A} = \{[\mathsf{attr}_{\mathsf{idx}_j}]_{2^k}\}_{j=0}^{2^h-2}, \{[\mathsf{v}_j]_{2^k}\}_{j=0}^{2^h-2}, \{[\mathsf{cond}_j]_{2^k}\}_{j=0}^{2^h-2})$ // 1 rounds & $(2^h-1)\cdot 18k^2$ bits in offline, 10 rounds & $(2^h-1)\cdot(27k + (9k-9)\log_2(p') + 21)$ bits in online
7: $[\mathsf{leafVal}]_{2^k} \leftarrow \pi_{\mathsf{rPathEval}}(\{[\mathsf{comp}_j]_2\}_{j=0}^{2^h-2}, \{[\mathsf{leafVal}_{j'}]_{2^k}\}_{j'=0}^{2^h-1}, \delta)$ // 4 rounds & $2^h(6k + 9h)$ bits
8: return $[\mathsf{leafVal}]_{2^k}$

Protocol 13 is our construction of PDTE over the ring. It employs Protocol 6 in the feature selection phase, Protocol 10 in the comparison phase, and Protocol 11 in the path evaluation phase.

4.6.2 Proposed Protocol of Private Decision Tree Evaluation with Semi-honest Security over Field

Protocol 14 $[\![\mathsf{leafVal}]\!]_p \leftarrow \pi_{\mathsf{pDTEval1}}(\{[\![\mathsf{idx}_j]\!]_p\}_{j=0}^{2^h-2}, \{[\![\mathsf{attr}_i]\!]_p\}_{i=0}^{m-1}, \{[\![\mathsf{v}_j]\!]_p\}_{j=0}^{2^h-2}, \{[\![\mathsf{cond}_j]\!]_p\}_{j=0}^{2^h-2}, \{[\![\mathsf{leafVal}_{j'}]\!]_p\}_{j'=0}^{2^h-1}, \delta)$

Input: Set of shared index number $\{[\![\mathsf{idx}_j]\!]_p\}_{j=0}^{2^h-2}$, shared feature array $\{[\![\mathsf{attr}_i]\!]_p\}_{i=0}^{m-1}$, threshold values $\{[\![\mathsf{v}_j]\!]_p\}_{j=0}^{2^h-2}$, conditional values $\{[\![\mathsf{cond}_j]\!]_p\}_{j=0}^{2^h-2}$, set of shared values assigned to leaf nodes $\{[\![\mathsf{leafVal}_{j'}]\!]_p\}_{j'=0}^{2^h-1}$, mapping function δ

Output: shared values assigned to leaf node of correct path $[\![\mathsf{leafVal}_{j'}]\!]_p$ where j' is s.t. $\bigwedge_{\ell=0}^{h-1}(j'|_\ell == \mathsf{comp}_{\delta(j',\ell)}) = 1$. Let $\mathsf{comp}_{\delta(j',\ell)}$ be $\mathsf{cond}_{\delta(j',\ell)} \cdot (\mathsf{v}_{\delta(j',\ell)} < \mathsf{attr}_{\mathsf{idx}_{\delta(j',\ell)}}) + (1 - \mathsf{cond}_{\delta(j',\ell)}) \cdot (\mathsf{v}_{\delta(j',\ell)} == \mathsf{attr}_{\mathsf{idx}_{\delta(j',\ell)}})$.

1: Initialize $\mathcal{A}$.
2: **for** $j = 0, \ldots, 2^h - 2$ **do**
3: (Feature Selection Phase)

4: $[\![\mathsf{attr}_{\mathsf{idx}_j}]\!]_p \leftarrow \pi_{\mathsf{pFSelection}}([\![\mathsf{idx}_j]\!]_p,\ \{[\![\mathsf{attr}_i]\!]_p\}_{i=0}^{m-1})$. // 3 rounds & $18m\log_2(p)$ bits in offline, 8 rounds & $27m\log_2(p)$ bits in online

5: Set $[\![\mathsf{attr}_{\mathsf{idx}_j}]\!]_p$ into $\mathcal{A}$.

6: (Comparison Phase)

7: $[\![\mathsf{v}_j < \mathsf{attr}_{\mathsf{idx}_j}]\!]_p \leftarrow \mathsf{pLTZ}(k, [\![\mathsf{v}_j - \mathsf{attr}_{\mathsf{idx}_j}]\!]_p)$ // 1 round & $(3k-3)\cdot 6\log_2(p)$ bits in offline and 3 rounds & $(k+1)\cdot 6\log_2(p)$ bits in online

8: $[\![\mathsf{v}_j == \mathsf{attr}_{\mathsf{idx}_j}]\!]_p \leftarrow \mathsf{pEQZ}(k, [\![\mathsf{v}_j - \mathsf{attr}_{\mathsf{idx}_j}]\!]_p)$ // 1 round & $(k - 3\log_2(k) - 2)\cdot 6\log_2(p)$ bits in offline and 3 rounds & $(\log_2(k)+2)\cdot 6\log_2(p)$ bits in online

9:

$$\begin{aligned} & [\![\mathsf{comp}_j]\!]_p \\ = \ & [\![\mathsf{cond}_j]\!]_p \cdot [\![\mathsf{v}_j < \mathsf{attr}_{\mathsf{idx}_j}]\!]_p \\ & +(1 - [\![\mathsf{cond}_j]\!]_p) \cdot [\![\mathsf{v}_j == \mathsf{attr}_{\mathsf{idx}_j}]\!]_p \\ = \ & [\![\mathsf{v}_j == \mathsf{attr}_{\mathsf{idx}_j}]\!]_p + [\![s_j]\!]_p \end{aligned}$$

where $[\![s_j]\!]_p \leftarrow \mathsf{pInnerProduct}(\{[\![\mathsf{cond}_j]\!]_p, [\![\mathsf{cond}_j]\!]_p\}, \{[\![\mathsf{v}_j < \mathsf{attr}_{\mathsf{idx}_j}]\!]_p, [\![\mathsf{v}_j == \mathsf{attr}_{\mathsf{idx}_j}]\!]_p\})$ // 1 round & $6\log_2(p)$ bits

10: **end for**

11: (Path Evaluation Phase)

12: Initialize $\mathsf{Path}_{j'} = \{[\![\mathsf{comp}_{\delta(j',0)}]\!]_p, [\![\mathsf{comp}_{\delta(j',1)}]\!]_p, \ldots, [\![\mathsf{comp}_{\delta(j',h-1)}]\!]_p\}$ by picking up $[\![\mathsf{comp}_{\delta(j',\ell)}]\!]_p$ corresponding to the intermediate value which has $(\ell+1)$-th height in the path to the $j^{'}$-th leaf node from $\{[\![\mathsf{comp}_j]\!]_p\}_{j=0}^{2^h-2}$.

13: **for** $j^{'} = 0, \ldots, 2^h - 1$ **do**

14: **for** $\ell = 0, \ldots, h-1$ **do**

15: $[\![c_{j',\ell}]\!]_p = [\![j^{'}|_\ell \oplus \mathsf{comp}_{\delta(j',\ell)}]\!]_p = (j^{'}|_\ell - [\![\mathsf{comp}_{\delta(j',\ell)}]\!]_p)^2 = j^{'}|_\ell + [\![\mathsf{comp}_{\delta(j',0)}]\!]_p - 2\cdot[\![\mathsf{comp}_{\delta(j',\ell)}]\!]_p$

16: **end for**

17: $[\![b_{j'}]\!]_p \leftarrow \mathsf{pArrayOr}(\{[\![c'_{j',\ell}]\!]_p\}_{\ell=0}^{h-1})$ // 1 round & $\log_2(h)\cdot 6\log_2(p)$ bits in offline and 3 rounds & $3\log_2(h)\cdot 6\log_2(p)$ bits in online

18: $[\![\mathsf{pathBit}_{j'}]\!]_p = [\![1 \oplus b_{j'}]\!]_p = (1 - [\![b_{j'}]\!]_p)^2 = 1 + [\![b_{j'}]\!]_p - 2\cdot[\![b_{j'}]\!]_p$

19: **end for**

20: return $[\![\mathsf{leafVal}]\!]_p \leftarrow \mathsf{pInnerProduct}(\{[\![\mathsf{pathBit}_{j'}]\!]_p\}_{j'=0}^{2^h-1}, \{[\![\mathsf{leafVal}_{j'}]\!]_p\}_{j'=0}^{2^h-1})$.

21: // 1 round & $6\log_2(p)$ bits

Protocol 15 $[\![\mathsf{leafVal}]\!]_p \leftarrow \pi_{\mathsf{pDTEval2}}(\{[\![\mathsf{idx}_j]\!]_p\}_{j=0}^{2^h-2}, \{[\![\mathsf{attr}_i]\!]_p\}_{i=0}^{m-1}, \{[\![\mathsf{v}_j]\!]_p\}_{j=0}^{2^h-2}, \{[\![\mathsf{cond}_j]\!]_p\}_{j=0}^{2^h-2}, \{[\![\mathsf{leafVal}_{j'}]\!]_p\}_{j'=0}^{2^h-1}, \delta)$

Input: Set of shared index number $\{[\![\mathsf{idx}_j]\!]_p\}_{j=0}^{2^h-2}$, shared feature array $\{[\![\mathsf{attr}_i]\!]_p\}_{i=0}^{m-1}$, threshold values $\{[\![\mathsf{v}_j]\!]_p\}_{j=0}^{2^h-2}$, conditional values $\{[\![\mathsf{cond}_j]\!]_p\}_{j=0}^{2^h-2}$, set of shared values assigned to leaf nodes $\{[\![\mathsf{leafVal}_{j'}]\!]_p\}_{j'=0}^{2^h-1}$, mapping function δ

Output: shared values assigned to leaf node of correct path $[\![\mathsf{leafVal}_{j'}]\!]_p$ where j' is s.t. $\bigwedge_{\ell=0}^{h-1}(j'|_\ell == \mathsf{comp}_{\delta(j',\ell)}) = 1$. Let $\mathsf{comp}_{\delta(j',\ell)}$ be $\mathsf{cond}_{\delta(j',\ell)} \cdot (\mathsf{v}_{\delta(j',\ell)} < \mathsf{attr}_{\mathsf{idx}_{\delta(j',\ell)}}) + (1 - \mathsf{cond}_{\delta(j',\ell)}) \cdot (\mathsf{v}_{\delta(j',\ell)} == \mathsf{attr}_{\mathsf{idx}_{\delta(j',\ell)}})$.

1: Initialize $\mathcal{A}$.
2: **for** $j = 0, \ldots, 2^h - 2$ **do**
3: (Feature Selection Phase)
4: $[\![\mathsf{attr}_{\mathsf{idx}_j}]\!]_p \leftarrow \pi_{\mathsf{pFSelection}}([\![\mathsf{idx}_j]\!]_p, \{[\![\mathsf{attr}_i]\!]_p\}_{i=0}^{m-1})$. // 3 rounds & $(2^h - 1) \cdot 18m \log_2(p)$ bits in offline, 8 rounds & $(2^h - 1) \cdot 27m \log_2(p)$ bits in online
5: Set $[\![\mathsf{attr}_{\mathsf{idx}_j}]\!]_p$ into $\mathcal{A}$.
6: (Comparison Phase)
7: $[\![\mathsf{v}_j < \mathsf{attr}_{\mathsf{idx}_j}]\!]_p \leftarrow \mathsf{pLTZ}(k, [\![\mathsf{v}_j - \mathsf{attr}_{\mathsf{idx}_j}]\!]_p)$ // 1 round & $(3k - 3) \cdot 6 \log_2(p)$ bits in offline and 3 round & $(k + 1) \cdot 6 \log_2(p)$ bits in online
8: $[\![\mathsf{v}_j == \mathsf{attr}_{\mathsf{idx}_j}]\!]_p \leftarrow \mathsf{pEQZ}(k, [\![\mathsf{v}_j - \mathsf{attr}_{\mathsf{idx}_j}]\!]_p)$ // 1 round & $(k - 3\log_2(k) - 2) \cdot 6 \log_2(p)$ bits in offline and 3 round & $(\log_2(k) + 2) \cdot 6 \log_2(p)$ bits in online
9:

$$\begin{aligned} & [\![\mathsf{comp}_j]\!]_p \\ = \; & [\![\mathsf{cond}_j]\!]_p \cdot [\![\mathsf{v}_j < \mathsf{attr}_{\mathsf{idx}_j}]\!]_p \\ & + (1 - [\![\mathsf{cond}_j]\!]_p) \cdot [\![\mathsf{v}_j == \mathsf{attr}_{\mathsf{idx}_j}]\!]_p \\ = \; & [\![\mathsf{v}_j == \mathsf{attr}_{\mathsf{idx}_j}]\!]_p + [\![s_j]\!]_p \end{aligned}$$

where $[\![s_j]\!]_p \leftarrow \mathsf{pInnerProduct}(\{[\![\mathsf{cond}_j]\!]_p, [\![\mathsf{cond}_j]\!]_p\}, \{[\![\mathsf{v}_j < \mathsf{attr}_{\mathsf{idx}_j}]\!]_p, [\![\mathsf{v}_j == \mathsf{attr}_{\mathsf{idx}_j}]\!]_p\})$ // 1 rounds & $6 \log_2(p)$ bits

10: **end for**
11: (Path Evaluation Phase)
12: $[\![\mathsf{leafVal}]\!]_p \leftarrow \pi_{\mathsf{pPathEval}}(\{[\![\mathsf{comp}_j]\!]_p\}_{j=0}^{2^h-2}, \{[\![\mathsf{leafVal}_{j'}]\!]_p\}_{j'=0}^{2^h-1}, \delta)$ // 3 rounds & $2^h \cdot 18 \log_2(p)$ bits in offline, 8 rounds & $2^h \cdot 27 \log_2(p)$ bits in online
13: return $[\![\mathsf{leafVal}]\!]_p$

Protocols 14 and 15 are the proposed constructions over the field. One of our schemes over the field (Protocol 14) is the same as the naive construction (Protocol 2 in 4.2.12), except that our scheme employs Protocol 7 in the feature selection phase. The other of our schemes over the field (Protocol 15) is the same as the naive construction (Protocol 2 in 4.2.12), except that our scheme employs Protocols 7 and 12 in the feature selection phase and the path evaluation phase, respectively.

4.7 Proposed Protocols with Fairness

Key Idea. To construct the maliciously secure PDTE protocol based only on SS-MPC with fairness and constant rounds, we use the fair 4PC based on secret sharing scheme,

Trident [12] and follow the algorithms of [22]. However, the PDTE protocol of [22] is based on semi-honest 3PC including the semi-honest shuffle protocol with constant rounds [62] and semi-honest private compare (PC) protocol with constant rounds [33][14]. The authors of Trident did not propose an oblivious shuffle protocol with constant rounds and PC protocol in [12]. Hence, it is non-trivial to realize the algorithms of the PDTE protocol [22] while achieving malicious security, fairness, and constant rounds by using Trident only straightforwardly.

To overcome this problem, we propose a maliciously secure shuffle protocol with fairness and constant rounds based on Trident. We also construct a PC protocol that achieves malicious security with fairness and constant rounds by using our shuffle as a subprotocol. By using our shuffle, PC, and Trident as building blocks, and modifying the algorithms of [22], we can realize the maliciously secure PDTE protocol with fairness and constant rounds.

4.7.1 Proposed Oblivious Shuffling Protocol

Protocol 16 Oblivious Mini-shuffle Protocol (Type 1) $\Pi_{\mathsf{miniShuffle1}}$

Input: Random permutation $\pi \in S_M$, senders (knowing π) $\{P_0, P_1, P_2\}$, receiver (not knowing π) P_3, the array of shares $[\vec{x}]_L = ([x_0]_L, [x_1]_L, \ldots, [x_{M-2}]_L, [x_{M-1}]_L)$

Output: Shuffled array of shares $[\vec{x'}]_L = ([x'_0]_L, [x'_1]_L, \ldots, [x'_{M-2}]_L, [x'_{M-1}]_L)$ where $x'_{\pi(\ell)} = x_\ell$ $(\ell = 0, \ldots, M-1)$.

1: Set $\mathsf{m}_{x_\ell} = x_\ell + \lambda_{x_\ell} \mod L$ and $\lambda_{x_\ell} = \lambda_{x_\ell,0} + \lambda_{x_\ell,1} + \lambda_{x_\ell,2} \mod L$ (for $\ell = 0, \ldots, M-1$).
2: Let $[x_\ell]_{L,3} = (\lambda_{x_\ell,0}, \lambda_{x_\ell,1}, \lambda_{x_\ell,2})$ be the P_3's shares of x_ℓ (for $\ell = 0, \ldots, M-1$).
3: Let $[x_\ell]_{L,i'} = (\mathsf{m}_{x_\ell}, \lambda_{x_\ell,\overline{i'+1}}, \lambda_{x_\ell,\overline{i'-1}})$ be the $P_{i'}$'s shares of x_ℓ where $i' \in \{0,1,2\}$ (for $\ell = 0, \ldots, M-1$).
4: **for** $\ell = 0, \ldots, M-1$ **do in parallel**
5: The senders P_0, P_1 and P_2 compute $r_{\ell,i'} = F_L(\mathsf{k}_{0,1,2}, \mathsf{uid}_{\ell,i'})$ and sets $r_\ell = r_{\ell,0} + r_{\ell,1} + r_{\ell,2} \mod L$ where $\mathsf{uid}_{\ell,i'}$ is a unique identifier and $i' \in \{0,1,2\}$.
6: $P_{i'}$ sets $[x'_{\pi(\ell)}]_{L,i'} = (\mathsf{m}_{x_\ell} + r_\ell \mod L, \lambda_{x_\ell,\overline{i'+1}} + r_{\ell,\overline{i'+1}} \mod L, \lambda_{x_\ell,\overline{i'-1}} + r_{\ell,\overline{i'-1}} \mod L)$.
7: **for** $i' = 0, 1, 2$ **do in parallel**
8: $P_{\overline{i'+1}}$ and $P_{\overline{i'-1}}$ compute $\lambda'_{\pi(\ell),i'} = \lambda_{x_\ell,i'} + r_{\ell,i'} \mod L$.
9: By $\mathsf{CC}(\{P_{\overline{i'+1}}, P_{\overline{i'-1}}\}, \lambda'_{\pi(\ell),i'}, P_3)$, P_3 gets $\lambda'_{\pi(\ell),i'}$ as the correct value $\lambda_{x'_{\pi(\ell)},i'}$ or aborts the protocol. `// 1 round & log2 L bits in offline, 0 round & 0 bit in online`
10: **end for**
11: **end for**
12: P_3 sets $[x'_{\pi(\ell)}]_{L,3} = (\lambda_{x'_{\pi(\ell)},0}, \lambda_{x'_{\pi(\ell)},1}, \lambda_{x'_{\pi(\ell)},2})$.

[14] In the client-server model [5], PC protocol [33] can achieve only privacy (not correctness) against a malicious adversary. However, we would like to construct the PDTE protocol achieving privacy and correctness against a malicious adversary even outside the client-server model. Hence, we cannot use the PC protocol [33] without modifications as a building block.

13: Return $[\vec{x'}]_L = ([x'_0]_L, [x'_1]_L, \ldots, [x'_{M-2}]_L, [x'_{M-1}]_L)$.

Protocol 17 Oblivious Table Shuffle Protocol $\Pi_{\mathsf{TableShuffle}}$

Input: The number of rows R, the number of columns C, the array of shares $[\vec{x}_0^{(0)}] = ([x_{0,0}^{(0)}]_{L_0}, \ldots, [x_{0,C-1}^{(0)}]_{L_{C-1}})$, …, $[\vec{x}_{R-1}^{(0)}] = ([x_{R-1,0}^{(0)}]_{L_0}, \ldots, [x_{R-1,C-1}^{(0)}]_{L_{C-1}})$ where modulus sizes $L_\ell \in \{2,\ 2^k, q\}$ $(\ell = 0, \ldots, C-1)$.

Output: Shuffled array of shares $[\vec{x}_0^{(4)}] = ([x_{0,0}^{(4)}]_{L_0}, \ldots, [x_{0,C-1}^{(4)}]_{L_{C-1}})$, …, $[\vec{x}_{R-1}^{(4)}] = ([x_{R-1,0}^{(4)}]_{L_0}, \ldots, [x_{R-1,C-1}^{(4)}]_{L_{C-1}})$ where $\pi \in S_R$ is a random permutation that no party knows and $x_{\pi(\ell'),\ell}^{(4)} = x_{\ell',\ell}^{(0)}$ for $\ell' = 0, \ldots, R-1;\ \ell = 0, \ldots, C-1$.

1: **for** $i = 0, 1, 2, 3$ **do**
2: P_i, $P_{\underline{i+1}}$, and $P_{\underline{i+2}}$ generate the random permutation $\pi_{\underline{i-1}} \in \mathcal{S}_R$ unknown to $P_{\underline{i-1}}$ by using the pseudo-random function F_R, the unique identifier uid_i, and pre-shared key $\mathsf{k}_{i,\underline{i+1},\underline{i+2}}$.
3: **end for**
4: Set $[\vec{y}_\ell^{(0)}] = ([y_{\ell,0}^{(0)}]_{L_\ell}, \ldots, [y_{\ell,R-1}^{(0)}]_{L_\ell}) = ([x_{0,\ell}^{(0)}]_{L_\ell}, \ldots, [x_{R-1,\ell}^{(0)}]_{L_\ell})$ for $\ell = 0, \ldots, C-1$.
5: **for** $\ell = 0, \ldots, C-1$ **do in parallel**
6: Parties obtain $[\vec{y}_\ell^{(1)}] = ([y_{\ell,0}^{(1)}]_{L_\ell}, [y_{\ell,1}^{(1)}]_{L_\ell}, \ldots, [y_{\ell,R-2}^{(1)}]_{L_\ell}, [y_{\ell,R-1}^{(1)}]_{L_\ell})$ by $\Pi_{\mathsf{miniShuffle1}}(\pi_3, \{P_0, P_1, P_2\}, P_3, [\vec{y}_\ell^{(0)}])$ where $y_{\pi_3(\ell')}^{(1)} = y_{\ell'}^{(0)}$ for $\ell' = 0, \ldots, R-1$.

// 1 round & $3R\log_2 L_\ell$ bits in offline, 0 round & 0 bit in online

7: **end for**
8: **for** $j = 1, 2, 3$ **do**
9: **for** $\ell = 0, \ldots, C-1$ **do in parallel**
10: Parties obtain $[\vec{y}_\ell^{(j+1)}] = ([y_{\ell,0}^{(j+1)}]_{L_\ell}, [y_{\ell,1}^{(j+1)}]_{L_\ell}, \ldots, [y_{\ell,R-2}^{(j+1)}]_{L_\ell}, [y_{\ell,R-1}^{(j+1)}]_{L_\ell})$ by $\Pi_{\mathsf{miniShuffle2}}(\pi_{j-1},\ \{P_3,\ P_{j-1},\ P_{\overline{j}}\},\ P_{\overline{j+1}},\ [\vec{y}_\ell^{(j)}])$ where $y_{\ell,\pi_{j-1}(\ell')}^{(j+1)} = y_{\ell,\ell'}^{(j)}$ for $\ell' = 0, \ldots, R-1$.

// 1 round & $2R\log_2 L_\ell$ bits in offline, 1 round & $R\log_2 L_\ell$ bits in online

11: **end for**
12: **end for**
13: Return $[\vec{x}_0^{(4)}] = ([x_{0,0}^{(4)}]_{L_0}, \ldots, [x_{0,C-1}^{(4)}]_{L_{C-1}}) = ([y_{0,0}^{(4)}]_{L_0}, \ldots, [y_{C-1,0}^{(4)}]_{L_{C-1}})$, $[\vec{x}_1^{(4)}] = ([x_{1,0}^{(4)}]_{L_0}, \ldots, [x_{1,C-1}^{(4)}]_{L_{C-1}}) = ([y_{0,1}^{(4)}]_{L_0}, \ldots, [y_{C-1,1}^{(4)}]_{L_{C-1}})$, …, $[\vec{x}_{R-1}^{(4)}] = ([x_{R-1,0}^{(4)}]_{L_0}, \ldots, [x_{R-1,C-1}^{(4)}]_{L_{C-1}}) = ([y_{0,R-1}^{(4)}]_{L_0}, \ldots, [y_{C-1,R-1}^{(4)}]_{L_{C-1}})$ where the random permutation $\pi = \pi_2 \circ \pi_1 \circ \pi_0 \circ \pi_3$ which no party knows and $x_{\pi(\ell'),\ell}^{(4)} = x_{\ell',\ell}^{(0)}$ for $\ell' = 0, \ldots, R-1;\ \ell = 0, \ldots, C-1$.

Overview. We propose the oblivious shuffle protocol for secret shares of table data (i.e., two-dimensional arrays) with fairness and constant rounds (Protocol 17, $\Pi_{\mathsf{TableShuffle}}$). To construct $\Pi_{\mathsf{TableShuffle}}$, we also propose the oblivious mini-shuffle protocol with fairness and constant rounds ($\Pi_{\mathsf{miniShuffle1}}$ and $\Pi_{\mathsf{miniShuffle2}}$). $\Pi_{\mathsf{miniShuffle1}}$ and $\Pi_{\mathsf{miniShuffle2}}$ take the random permutation that only three parties know and the shares of the one-dimensional array. They output the shares of the shuffled array with fairness by resharing the three parties' (locally shuffled) shares via CC. Since $\Pi_{\mathsf{miniShuffle1}}$ and $\Pi_{\mathsf{miniShuffle2}}$ consist of the local operations

and execution of CC, they can achieve fairness and constant rounds. $\Pi_{\mathsf{TableShuffle}}$ can be constructed by executing the oblivious mini-shuffles in series while changing the random permutation and the three parties who know the permutation. Hence, $\Pi_{\mathsf{TableShuffle}}$ achieves fairness and constant rounds.

Our shuffle protocols follow the high-level strategy of resharing-based shuffle and can be regarded as a kind of constant-round resharing-based shuffle among three parties [62, 68, 69] or multi-party resharing-based shuffle [67]. However, we emphasize that the existing constant-round resharing-based shuffle protocols among three parties [62, 68, 69] are only semi-honest secure. We also emphasize that the existing multi-party resharing-based shuffle protocol with the malicious security [67] can achieve the security only with abort by using zero-knowledge proofs and commitment scheme, without fairness. In addition, it can detect cheating probabilistically. That is, if the parties would like to increase the probability of cheating detection, parties need to sacrifice performance. On the other hand, our shuffle protocol can detect cheating deterministically rather than probabilistically without sacrificing performance.

Intuition of Protocol 16. In $\Pi_{\mathsf{miniShuffle1}}$, P_0, P_1, and P_2 know the random permutation (and P_3 does not know it). In $\Pi_{\mathsf{miniShuffle2}}$, P_3, $P_{\overline{i'-1}}$, and $P_{\overline{i'+1}}$ know the random permutation (and $P_{i'}$ does not know it), where $i' \in \{0,1,2\}$.

In $\Pi_{\mathsf{miniShuffle1}}$, at Step 5, the three senders P_0, P_1, and P_2, compute the randomnesses $r_{\ell,i'}$ (such that $r_\ell = r_{\ell,0} + r_{\ell,1} + r_{\ell,2} \mod L$) by using $\mathsf{k}_{0,1,2}$ that P_3 does not know to rerandomize the shares of input array at next step (for $\ell = 0, \ldots, M-1; i' = 0,1,2$). At Step 6, the sender $P_{i'}$ ($i' = 0,1,2$) shuffles m_{x_ℓ}, $\lambda_{x_\ell,\overline{i'+1}}$ and $\lambda_{x_\ell,\overline{i'-1}}$ locally by applying π and rerandomizes the (locally) shuffled values by using $r_\ell, r_{\ell,0}, r_{\ell,1}$, and $r_{\ell,2}$ and setting $[x'_{\pi(\ell)}]_{L,i'} = (\mathsf{m}_{x_\ell} + r_\ell \mod L, \lambda_{x_\ell,\overline{i'+1}} + r_{\ell,\overline{i'+1}} \mod L, \lambda_{x_\ell,\overline{i'-1}} + r_{\ell,\overline{i'-1}} \mod L)$. At Steps 7 to 10, the senders send the (locally) shuffled and rerandomized values $\lambda'_{\pi(\ell),i'}$ to the receiver P_3 by CC. Note that $\lambda'_{\pi(\ell),i'}$ leaks no information about the permutation π and the values before the shuffle because of rerandomizing it by $r_\ell, r_{\ell,0}, r_{\ell,1}, r_{\ell,2}$. Then, the receiver P_3 constructs the shuffled shares $[x'_{\pi(\ell)}]_{L,3}$ without knowing π and the shares before the shuffle. $\Pi_{\mathsf{miniShuffle2}}$ and $\Pi_{\mathsf{miniShuffle1}}$ are almost identical except that the senders and receiver are different.

Note that P_3's shares are independent of the actual array. Hence, $\Pi_{\mathsf{miniShuffle1}}$ (and $\Pi_{\mathsf{miniShuffle2}}$) allow the resharing values related to P_3's shares by CC to be processed in the offline phase as well as in Trident.

To explain in more detail, three senders, P_0, P_1 and P_2, can send the shuffled three values $\lambda'_{\pi(\ell),i'}$ (for $i' = 0,1,2$) to the receiver, P_3 in the offline phase at Step 9 in $\Pi_{\mathsf{miniShuffle1}}$. On the other hand, in $\Pi_{\mathsf{miniShuffle2}}$, three senders, P_3, $P_{\overline{i'+1}}$ and $P_{\overline{i'-1}}$, can send the shuffled two values, $\lambda'_{\pi(\ell),\overline{i'+1}}$ and $\lambda'_{\pi(\ell),\overline{i'-1}}$, to the receiver, $P_{i'}$ in the offline phase. The remaining value of shuffled shares of $P_{i'}$, $\mathsf{m}_{x'_{\pi(\ell)}}$ must be sent to $P_{i'}$ in the online phase by three senders because $\mathsf{m}_{x'_{\pi(\ell)}}$ depends on the actual array. For this reason, the communication costs in the offline/online phase of $\Pi_{\mathsf{miniShuffle2}}$ differ from that of $\Pi_{\mathsf{miniShuffle1}}$.

Intuition of Protocol 17. $\Pi_{\mathsf{TableShuffle}}$ takes the matrix of shares with R rows and C columns and outputs the matrix with the shuffled rows. In $\Pi_{\mathsf{TableShuffle}}$, the three parties

generate the random permutation that the rest of the parties does not know by F_R, a unique identifier, and pre-shared keys from Steps 1 to 3. At Step 4, the parties set the column vector using input shares. From Steps 5 to 12, the parties run $\Pi_{\mathsf{miniShuffle1}}$ and $\Pi_{\mathsf{miniShuffle2}}$ for each column vector in series while changing the random permutation and the senders who know the random permutation. Then, the parties set the shuffled rows using the shuffled columns at Step 13.

4.7.2 Proposed MSB Extraction, LT, and EQ Protocols

Protocol 18 Random shares generation protocol over $(2,4)$-RSS Π_{RSG}

Input: Modulus size L

Output: $[r]_L$ s.t. $r \in \mathbb{Z}_L$ which no party knows. (Note that r is generated rondomly.)

1: P_3, $P_{\overline{i'-1}}$, and $P_{\overline{i'+1}}$ compute $\lambda_{r,i'} = F_L(\mathsf{uid}_{\mathsf{RSG}}, \mathsf{k}_{3,\overline{i'-1},\overline{i'+1}})$ for $i' = 0,1,2$ where $\mathsf{uid}_{\mathsf{RSG}}$ is a unique identifier.
2: P_0, P_1, and P_2 computes $\mathsf{m}_r = F_L(\mathsf{uid}_{\mathsf{RSG}}, \mathsf{k}_{0,1,2})$ and set $\mathsf{m}_r = r + \lambda_r \mod L$ and $\lambda_r = \lambda_{r,0} + \lambda_{r,1} + \lambda_{r,2} \mod L$.
3: P_3 sets $[r]_{L,3} = (\lambda_{r,0}, \lambda_{r,1}, \lambda_{r,2})$. $P_{i'}$ sets $[r]_{L,i'} = (\mathsf{m}_r, \lambda_{r,\overline{i'+1}}, \lambda_{r,\overline{i'-1}})$ for $i' = 0,1,2$ where $\mathsf{m}_r = r + \lambda_{r,0} + \lambda_{r,1} + \lambda_{r,2} \mod L$.
4: Return $[r]_L = ([r]_{L,0}, [r]_{L,1}, [r]_{L,2}, [r]_{L,3})$.

Protocol 19 Fair and Private Compare Protocol for k'-bit values Π_{FPC}

Input: Bit length $k'(\leq k)$, binary shares $\{[x|_\ell]_{p'}\}_{\ell=0}^{k'-1}$ (where $x|_\ell \in \{0,1\}$), a common input $r \in \{0,1\}^{k'}$

Output: $[(x > r)]_{2^k}$

1: (Offline phase)
2: **for** $\ell = 0, \ldots, k'-1$ **do in parallel**
3: P_3, $P_{\overline{i'-1}}$ and $P_{\overline{i'+1}}$ computes $s_{\ell,i'} = F^*_{p'}(\mathsf{uid}_\ell, \mathsf{k}_{3,\overline{i'-1},\overline{i'+1}})$ and $s'_{\ell,i'} = F^*_{p'}(\mathsf{uid}'_\ell, \mathsf{k}_{3,\overline{i'-1},\overline{i'+1}})$ for $i' = 0,1,2$ where uid_ℓ and uid'_ℓ are unique identifiers.
4: P_0, P_1 and P_2 computes $s_{\ell,3} = F^*_{p'}(\mathsf{uid}_\ell, \mathsf{k}_{0,1,2})$ and $s'_{\ell,3} = F^*_{p'}(\mathsf{uid}'_\ell, \mathsf{k}_{0,1,2})$.
5: Set $[s_{\ell,3}]_{p',3} = (0,0,0)$ and $[s_{\ell,3}]_{p',i'} = (s_{\ell,3}, 0, 0)$ for $i' = 0,1,2$.
6: Set $[s_{\ell,0}]_{p',0} = (0,0,0)$, $[s_{\ell,0}]_{p',1} = (0,0,-s_{\ell,0})$, $[s_{\ell,0}]_{p',2} = (0,-s_{\ell,0},0)$, and $[s_{\ell,0}]_{p',3} = (-s_{\ell,0},0,0)$.
7: Set $[s_{\ell,1}]_{p',0} = (0,-s_{\ell,1},0)$, $[s_{\ell,1}]_{p',1} = (0,0,0)$, $[s_{\ell,1}]_{p',2} = (0,0,-s_{\ell,1})$, and $[s_{\ell,1}]_{p',3} = (0,-s_{\ell,1},0)$.
8: Set $[s_{\ell,2}]_{p',0} = (0,0,-s_{\ell,2})$, $[s_{\ell,2}]_{p',1} = (0,-s_{\ell,2},0)$, $[s_{\ell,2}]_{p',2} = (0,0,0)$, and $[s_{\ell,2}]_{p',3} = (0,0,-s_{\ell,2})$.
9: Set $[s'_{\ell,i}]_{p'}$ in the same way as $[s_{\ell,i}]_{p'}$ by using $s'_{\ell,i}$ for $i = 0,\ldots,3$.
10: Each party computes $[s_{\ell,0} \cdot s_{\ell,1}]_{p'}$, $[s_{\ell,2} \cdot s_{\ell,3}]_{p'}$, $[s'_{\ell,0} \cdot s'_{\ell,1}]_{p'}$ and $[s'_{\ell,2} \cdot s'_{\ell,3}]_{p'}$ by $\mathsf{Mult}([s_{\ell,0}]_{p'}, [s_{\ell,1}]_{p'})$, $\mathsf{Mult}([s_{\ell,2}]_{p'}, [s_{\ell,3}]_{p'})$, $\mathsf{Mult}([s'_{\ell,0}]_{p'}, [s'_{\ell,1}]_{p'})$ and $\mathsf{Mult}([s'_{\ell,2}]_{p'}, [s'_{\ell,3}]_{p'})$, respectively. `// 2 rounds & 24 log2 p' bits in offline, 0 round & 0 bit in online`
11: Each party computes $[s_\ell]_{p'} = [(s_{\ell,0} \cdot s_{\ell,1}) \cdot (s_{\ell,2} \cdot s_{\ell,3})]_{p'}$ and $[s'_\ell]_{p'} = [(s'_{\ell,0} \cdot s'_{\ell,1}) \cdot (s'_{\ell,2} \cdot$

$s'_{\ell,3})]_{p'}$ by $\mathsf{Mult}([s_{\ell,0}\cdot s_{\ell,1}]_{p'}, [s_{\ell,2}\cdot s_{\ell,3}]_{p'})$ and $\mathsf{Mult}([s'_{\ell,0}\cdot s'_{\ell,1}]_{p'}, [s'_{\ell,2}\cdot s'_{\ell,3}]_{p'})$, respectively. // 2 rounds & $12\log_2 p'$ bits in offline, 0 round & 0 bit in online

end for

Parties compute $[b]_2$ by $\Pi_{\mathsf{RSG}}(2)$.

Parties get the shares of random bit $[b]_{2^k}$ and $[b]_{p'}$ by $\mathsf{BitConv}(2^k, [b]_2)$ and $\mathsf{BitConv}(p', [b]_2)$, respectively. // 3 rounds & $6k + 6\log_2 p' + 2$ bits in offline, 0 round & 0 bit in online

(Online phase)

Let $t = r + 1 \mod 2^k$.

for $\ell = k' - 1, \ldots, 0$ **do in parallel**

(Case of $b = 0$)

$[w|_\ell]_{p'} = [x|_\ell]_{p'} + r|_\ell - 2r|_\ell[x|_\ell]_{p'}$, $[c|_\ell]_{p'} = r|_\ell - [x|_\ell]_{p'} + 1 + \sum_{m=\ell+1}^{k-1}[w|_m]_{p'}$

(Case of $b = 1$)

$[w'|_\ell]_{p'} = [x|_\ell]_{p'} + t|_\ell - 2t|_\ell[x|_\ell]_{p'}$, $[c'|_\ell]_{p'} = -t|_\ell + [x|_\ell]_{p'} + 1 + \sum_{m=\ell+1}^{k-1}[w'|_m]_{p'}$

end for

$[s_\ell \cdot c|_\ell]_{p'} \leftarrow \mathsf{Mult}([s_\ell]_{p'}, [c|_\ell]_{p'})$ and $[s'_\ell \cdot c'|_\ell]_{p'} \leftarrow \mathsf{Mult}([s'_\ell]_{p'}, [c'|_\ell]_{p'})$ for $\ell = 0, \ldots, k' - 1$ in parallel. // 1 round & $6k'\log_2 p'$ bits in offline, 1 round & $6k'\log_2 p'$ bits in online

Parties get the shuffled array $[\vec{d}]_{p'} = ([d_0]_{p'}, \ldots, [d_{k'-1}]_{p'})$ and $[\vec{d'}]_{p'} = ([d'_0]_{p'}, \ldots, [d'_{k'-1}]_{p'})$ by $\Pi_{\mathsf{TableShuffle}}(k', 1, [s_0 \cdot c|_0]_{p'}, \ldots, [s_{k'-1} \cdot c|_{k'-1}]_{p'})$ and $\Pi_{\mathsf{TableShuffle}}(k', 1, [s'_0 \cdot c'|_0]_{p'}, \ldots, [s'_{k'-1} \cdot c'|_{k'-1}]_{p'})$ in parallel, respectively. // 4 rounds & $18k'\log_2 p'$ bits in offline, 3 rounds & $6k'\log_2 p'$ bits in online

for $\ell = k' - 1, \ldots, 0$ **do in parallel**

$[d''_\ell]_{p'} = (1 - [b]_{p'}) \cdot [d_\ell]_{p'} + [b]_{p'} \cdot [d'_\ell]_{p'} = [d_\ell]_{p'} + [b]_{p'} \cdot (-[d_\ell]_{p'} + [d'_\ell]_{p'}) = [d_\ell]_{p'} + \mathsf{Mult}([b]_{p'}, [-d_\ell + d'_\ell]_{p'})$. // 1 round & $3\log_2 p'$ bits in offline, 1 round & $3\log_2 p'$ bits in online

for $i' = 0, 1, 2$ **do in parallel**

$P_{i'}$ reconstructs d''_ℓ by $\mathsf{OpenOne}(P_{i'}, [d''_\ell]_{p'})$. // 0 round & 0 bit in offline, 1 round & $\log_2 p'$ bits in online

end for

end for

P_0, P_1, and P_2 set $b' = 1$ iff $\exists \ell \in \{0, \ldots, k' - 1\}$ s.t. $d''_\ell = 0$ else $b' = 0$.

All parties computes $\lambda_{b',i'} = F_{2^k}(\mathsf{uid}_{b'}, \mathsf{k})$ for $i' = 0, 1, 2$.

P_3 sets $[b']_{2^k,3} = (\lambda_{b',0}, \lambda_{b',1}, \lambda_{b',2})$. $P_{i'}$ sets $[b']_{2^k,i'} = (\mathsf{m}_{b'}, \lambda_{b',\overline{i'+1}}, \lambda_{b',\overline{i'-1}})$ for $i' = 0, 1, 2$ where $\mathsf{m}_{b'} = b' + \sum_{i'=0}^{2} \lambda_{b',i'} \mod 2^k$.

Return $[(x > r)]_{2^k} = [b' \oplus b]_{2^k} = ([b']_{2^k} - [b]_{2^k})^2 \leftarrow \mathsf{Mult}([b' - b]_{2^k}, [b' - b]_{2^k})$. // 1 round & $3k$ bits in offline, 1 round & $3k$ bits in online

---

Protocol 20 Most Significant Bit Extraction Protocol Π_{msbExt}

Input: $[x]_{2^k}$ s.t. $x \in \mathbb{Z}_{2^k}, x = \sum_{j=0}^{k-1} 2^j \cdot x|_j$

Output: $[\mathsf{msb}(x)]_2 (= [x|_{k-1}]_2)$

(Offline phase)
Parties compute $[r|_\ell]_2$ by $\Pi_{\mathsf{RSG}}(2)$ for $\ell = 0, \ldots, k-1$.
for $j = 0, \ldots, k-1$ **do in parallel**
  Parties get $[r|_j]_{2^k}$ and $[r|_j]_{p'}$ by $\mathsf{BitConv}(2^k, [r|_j]_2)$ and $\mathsf{BitConv}(p', [r|_j]_2)$ in parallel, respectively.
  `// 3 rounds & 6k + 6 log_2 p' + 2 bits in offline, 0 round & 0 bit in online`
end for
$[r|_{k-2,\ldots,0}]_{2^k} = \sum_{j=0}^{k-2} 2^j \cdot [r|_j]_{2^k}$, $[2^{k-1} \cdot \mathsf{msb}(r)]_{2^k} = 2^{k-1} \cdot [r|_{k-1}]_{2^k}$.
(Online phase)
$[x + (r|_{k-2,\ldots,0})]_{2^k} = [x]_{2^k} + [r|_{k-2,\ldots,0}]_{2^k}$
$[2 \cdot ((x+r)|_{k-2,\ldots,0})]_{2^k} = 2 \cdot [x + (r|_{k-2,\ldots,0})]_{2^k}$
$P_{i'}$ reconstructs $2 \cdot ((x+r)|_{k-2,\ldots,0})$ by $\mathsf{OpenOne}(P_{i'}, [2 \cdot ((x+r)|_{k-2,\ldots,0})]_{2^k})$ for $i' = 0, 1, 2$ in parallel. `// 0 round & 0 bit in offline, 1 round & 3k bits in online`
P_3 sets $[(x+r)|_{k-2,\ldots,0}]_{2^k,3} = (0, 0, 0)$.
$P_{i'}$ sets $[(x+r)|_{k-2,\ldots,0}]_{2^k,i'} = ((x+r)|_{k-2,\ldots,0}, 0, 0)$ for $i' = 0, 1, 2$.
If $(x+r)|_{k-2,\ldots,0} = 2^{k-1} - 1$, P_0, P_1, and P_2 set the bit $\mathsf{needFPC} = 0$. If not, they set $\mathsf{needFPC} = 1$. Then, P_0 and P_1 send $\mathsf{needFPC}$ to P_3 by CC. `// 0 round & 0 bit in offline, 1 round & 1 bit in online`
If $\mathsf{needFPC} = 0$, $[x|_{k-2,\ldots,0}]_{2^k} = [(x+r)|_{k-2,\ldots,0}]_{2^k} - [r|_{k-2,\ldots,0}]_{2^k}$.
If $\mathsf{needFPC} = 1$, $[r|_{k-2,\ldots,0} > (x+r)|_{k-2,\ldots,0}]_{2^k} \leftarrow \Pi_{\mathsf{FPC}}(k-1, \{[r|_\ell]_{p'}\}_{\ell=0}^{k-2}, (x+r)|_{k-2,\ldots,0})$
`// 14 rounds & (63k − 57) log_2 p' + 9k + 2 bits in offline, 7 rounds & 18(k − 1) log_2 p' + 3k bits in online`
If $\mathsf{needFPC} = 1$, $[x|_{k-2,\ldots,0}]_{2^k} = [(x+r)|_{k-2,\ldots,0}]_{2^k} - [r|_{k-2,\ldots,0}]_{2^k} + 2^{k-1} \cdot [r|_{k-2,\ldots,0} > (x+r)|_{k-2,\ldots,0}]_{2^k}$.
$[2^{k-1} \cdot \mathsf{msb}(x)]_{2^k} = [2^{k-1} \cdot x|_{k-1}] = [x]_{2^k} - [x|_{k-2,\ldots,0}]_{2^k}$
$[2^{k-1} \cdot (\mathsf{msb}(x) \oplus \mathsf{msb}(r))]_{2^k} = [2^{k-1} \cdot \mathsf{msb}(x)]_{2^k} + [2^{k-1} \cdot \mathsf{msb}(r)]_{2^k} = 2^{k-1} \cdot [x|_{k-1}]_{2^k} + 2^{k-1} \cdot [r|_{k-1}]_{2^k}$
for $i' = 0, 1, 2$ **do in parallel**
  $P_{i'}$ gets $2^{k-1} \cdot (\mathsf{msb}(x) \oplus \mathsf{msb}(r))$ by $\mathsf{OpenOne}(P_{i'}, [2^{k-1} \cdot (\mathsf{msb}(x) \oplus \mathsf{msb}(r))]_{2^k})$
  `// 0 round & 0 bit in offline, 1 round & 3k bits in online`
end for
P_3 sets $[\mathsf{msb}(x) \oplus \mathsf{msb}(r)]_{2,3} = (0, 0, 0)$.
$P_{i'}$ sets $[\mathsf{msb}(x) \oplus \mathsf{msb}(r)]_{2,i'} = (\mathsf{msb}(x) \oplus \mathsf{msb}(r), 0, 0)$ for $i' = 0, 1, 2$.
Return $[\mathsf{msb}(x)]_2 = [\mathsf{msb}(x) \oplus \mathsf{msb}(r)]_2 \oplus [r|_{k-1}]_2$

Overview. We propose the PC protocol for k'-bit values achieving malicious security with fairness and constant rounds (Protocol 19, Π_{FPC}). Π_{FPC} takes the bit length k', binary shares $\{[x|_\ell]_{p'}\}_{\ell=0}^{k'-1}$, and common input r and outputs $[x > r]_{2^k}$.

To construct Π_{FPC}, we follow the high-level strategy of PC protocol of SecureNN [33]. However, the existing constant-round PC protocol [33] achieves only the semi-honest security. In addition, the existing constant-round PC protocol [33] is an asymmetric 3PC protocol. Most of asymmetric MPC protocols are designed to reduce the communication

complexities by sacrificing the achievement of a high level of security. Thus, in general, it seems harder for the maliciously secure (symmetric) MPC to reduce the communication complexities or to be the constant-round protocol.

Hence, just using the existing maliciously secure MPC protocol and following the high-level strategy of [33] straightforwardly do not seem to lead to the protocol with fairness and constant rounds. In particular, the existing constant-round PC protocol [33] uses the (non-zero) random value generation and asymmetric oblivious shuffle. The (non-zero) random value generation is more difficult to achieve in the setting of maliciously secure (symmetric) MPC than in that of semi-honest secure (asymmetric) MPC. The intuitive reason why it is difficult is that it requires all parties to work together to generate a non-zero random number that is unknown to all parties, while it also requires a technique to detect whether a party is dishonestly using zero as a random number. The technique to detect cheating is not required in that of semi-honest secure (asymmetric) MPC because parties follow the protocol specification. In addition, the existing maliciously secure shuffle [67] achieves security with only abort, without fairness.

To overcome the above and construct the maliciously secure PC protocol with fairness and constant rounds, we use our new maliciously secure shuffle $\Pi_{\mathsf{TableShuffle}}$ and convert the asymmetric semi-honest secure 3PC protocol [33] to the symmetric maliciously secure 4PC protocol based on Trident.

Then, we construct the maliciously secure MSB extraction protocol with fairness and constant rounds (Protocol 20, Π_{msbExt}) by following the high-level strategy of the existing constant-round MSB extraction protocol [22] but introducing our new secure PC protocol Π_{FPC} and converting the semi-honest secure 3PC protocol [22] to the maliciously secure 4PC protocol based on Trident. Note that the maliciously secure PC protocol with fairness using the fair MPC and following the high-level strategy of [22] straightforwardly requires the computing the greater-than circuit and the number of communication rounds proportional to the size of the circuit.

Secure LT (Π_{LT}) and EQ (Π_{EQ}) protocols can be realized by using fair MPC and following the high-level strategy of [22] straightforwardly, but then we require computing the circuit for MSB extraction and the number of communication rounds proportional to the size of the circuit. We avoid such realizations and instantiate the maliciously secure LT (Π_{LT}) and EQ (Π_{EQ}) protocols by following the high-level strategy of existing constant-round protocols [22] but using our new introduced Π_{msbExt}.

Intuition of Protocol 18. Π_{RSG} takes the modulus size L as input and output the shares of the random number that no party knows, $[r]_L$. This protocol is used as a subprotocol in Protocols 19 and 20.

At Steps 1 and 2, three parties compute each value that the three parties hold over $(2,4)$-RSS by using the pseudo-random function, the unique identifier, and the pre-shared key that the three parties hold without communications. For example, at Step 1, P_3, $P_{\overline{i'-1}}$ and $P_{\overline{i'+1}}$ compute $\mathsf{m}_r \in \mathbb{Z}_L$ by using F_L, $\mathsf{uid}_{\mathsf{RSG}}$, and $\mathsf{k}_{3,\overline{i'-1},\overline{i'+1}}$. Here, $\mathsf{uid}_{\mathsf{RSG}} \in \{0,1\}^\kappa$ is a unique identifier. That is, $\mathsf{uid}_{\mathsf{RSG}}$ is different each time Π_{RSG} is executed. We note that $P_{i'}$ cannot know $\lambda_{r,i'}$ because $P_{i'}$ does not hold $\mathsf{k}_{3,\overline{i'-1},\overline{i'+1}}$ for $i' = 0,1,2$. P_3 cannot know m_r because P_3 does not hold $\mathsf{k}_{0,1,2}$.

Then, parties let $r = \mathsf{m}_r + \lambda_r \mod L$ where $\lambda_r = \lambda_{r,0} + \lambda_{r,1} + \lambda_{r,2} \mod L$ at Step 3. We note that no party knows $r \in \mathbb{Z}_L$ because each party knows only three of the four values, m_r, $\lambda_{r,0}$, $\lambda_{r,1}$ and $\lambda_{r,2}$. After that, each party sets his/her shares by using three of the four values that he/she knows.

Intuition of Protocol 19. In the offline phase of Π_{FPC}, from Steps 2 to 12, parties compute the shares of the non-zero random values $s_\ell, s'_\ell \in \mathbb{F}^*_{p'}$ that no party knows. At Step 13, parties generate the shares of a random bit b over $\mathbb{Z}_2$ by Π_{RSG}. Then, the parties convert them into the shares over $\mathbb{Z}_{2^k}$ and $\mathbb{Z}_{p'}$ by $\mathsf{BitConv}$ at Step 14.

The strategy in the online phase of Π_{FPC} is almost the same as SecureNN [33]. That is, the parties compute the masked comparison result bit $[b \oplus (x > r)]_{2^k}$ and remove the mask b. The difference between SecureNN and our protocol is that b is shared by all parties and no party knows b. Therefore, the parties compute both the cases of $b = 0$ and $b = 1$ to compute $[b \oplus (x > r)]_2$. In other words, the parties compute both $(x > r)$ and $(x \leq r) \equiv (x < t)$ (where $t = r+1$) obliviously. Then, the parties do the oblivious selection by $[b]_2$ and remove it.

We focus on the explanation of the case of $b = 0$, i.e., the case of $[x > r]_{2^k}$. Note that it holds that $(x > r) = 1$ if and only if there exists the leftmost[15] ℓ'-th bit where $x|_{\ell'} \neq r|_{\ell'}$ and $x|_{\ell'} = 1$. The existence of such the ℓ'-th bit implies that the bits of x and r existing on left side of the ℓ'-th bit are identical. The parties compute $[w|_\ell]_{p'}$ and $[c|_\ell]_{p'}$ (at Step 19). Then, there exists the ℓ-th bit such that $c|_\ell = 0$ if $(x > r) = 1$. After that, at Step 23, the parties compute the masked shares of $c|_\ell$, $[s_\ell \cdot c|_\ell]_{p'}$ by using the shares of non-zero random value s_ℓ (computed in the offline phase). The parties obtain the shuffled array $[\vec{d}]_{p'}$ by $\Pi_{\mathsf{TableShuffle}}(k',\ 1,\ [s_0 \cdot c|_0]_{p'}, \ldots, [s_{k'-1} \cdot c|_{k'-1}]_{p'})$ at Step 24. The case of $b = 1$ is the same as that of $b = 0$ and is described at Steps 21, 23, and 24.

Then, the parties choose either $[\vec{d}]_{p'}$ or $[\vec{d'}]_{p'}$ as $[d''_\ell]_{p'}$ obliviously depending on the value of b at Step 26. After that, P_0, P_1, and P_2 reconstruct d''_ℓ by $\mathsf{OpenOne}$ from Steps 27 to 29. After reconstruction, at Step 31, if there exists 0 in $d''_0, \ldots, d''_{k'-1}$ (i.e., $\exists \ell \in \{0, \ldots, k'-1\}$ s.t. $d''_\ell = 0$), P_0, P_1, and P_2 set $b' = 1$. If not, they set $b' = 0$. The existence of 0 in $d''_0, \ldots, d''_{k'-1}$ means the existence of 0 in $d_0, \ldots, d_{k'-1}$ or $d'_0, \ldots, d'_{k'-1}$ depending on $b = 0$ or 1, respectively. It also means the existence of 0 in $c_0, \ldots, c_{k'-1}$ or $c'_0, \ldots, c'_{k'-1}$ depending on $b = 0$ or 1, respectively. Note that P_0, P_1, and P_2 cannot learn any new information about whether there exists 0 in $d''_0, \ldots, d''_{k'-1}$ because d''_ℓ is masked by the non-zero random value s_ℓ or s'_ℓ (that no party knows) and shuffled by $\Pi_{\mathsf{TableShuffle}}$. That is, $\{d_{\pi(\ell)} (= s_\ell \cdot c|_\ell)\}_{\ell=0}^{k'}$ and $\{d'_{\pi(\ell)} (= s'_\ell \cdot c'|_\ell)\}_{\ell=1}^{k'}$ where π is a random permutation that no one knows and is hiding the positional information on whether $\{c_\ell\}_{\ell=1}^{k'}$ and $\{c'_\ell\}_{\ell=1}^{k'}$ contain 0, respectively. $\{d''_\ell\}_{\ell=1}^{k'}$ hide whether $b = 0$ or 1. Hence, the reconstructed values $\{d''_\ell\}_{\ell=1}^{k'}$ do not leak the positional information on whether $\{c_\ell\}_{\ell=1}^{k'}$ or $\{c'_\ell\}_{\ell=1}^{k'}$ contain 0. At Steps 32 and 33, all parties set $[b']_{2^k}$ by using only local operations. After that, they compute $[x > r]_{2^k} = [b' \oplus b]_{2^k}$ by Mult at Step 34.

Intuition of Protocol 20. Π_{msbExt} takes $[x]_{2^k}$ and outputs $[\mathsf{msb}(x)]_2 = [x|_{k-1}]_2$. In the offline phase of Π_{msbExt}, the parties generate the shares of random values to mask the

[15] In this paper, the most significant bit is the leftmost bit and the least significant bit is the rightmost bit. For example, the most significant bit of $x \in \mathbb{Z}_{2^k}$, $x|_{k-1}$, means the leftmost $(k-1)$-th bit in x.

values of the calculation process. At Steps 2 to 5, the parties generate the shares of random value $r|_\ell \in \{0,1\}$ over $\mathbb{Z}_2$ and convert it into the shares over $\mathbb{Z}_{2^k}$ and $\mathbb{Z}_p$. After that, they compute the shares $[r|_{k-2,\ldots,0}]_{2^k}$ and $[2^{k-1} \cdot \mathsf{msb}(r)]_{2^k}$ at Step 6.

In the online phase of Π_{msbExt}, the first goal is to compute the shares $[x|_{k-2,\ldots,0}]_{2^k}$. To compute them, the parties compute $[2 \cdot ((x+r)|_{k-2,\ldots,0})]_{2^k}$ (at Steps 8 and 9). Then, P_0, P_1, and P_2 get $(x+r)|_{k-2,\ldots,0}$ by $\mathsf{OpenOne}$ at Step 10. The parties set $[(x+r)|_{k-2,\ldots,0}]_{2^k}$ at Steps 11 and 12. If $(x+r)|_{k-2,\ldots,0} = 2^{k-1}-1$, the parties set the flag bit $\mathsf{needFPC} = 0$, otherwise, they set $\mathsf{needFPC} = 1$ at Step 13. The bit $\mathsf{needFPC}$ means whether or not to run Π_{FPC} to cancel the effect of the wrap-around. The wrap-around means that the modulo operation may have $(x+r)|_{k-2,\ldots,0} \mod 2^k$ less than $r|_{k-2,\ldots,0} \mod 2^k$. If $\mathsf{needFPC} = 0$, i.e., $(x+r)|_{k-2,\ldots,0} = 2^{k-1}-1$, the wrap-around does not occur. The parties remove the shared mask $[r|_{k-2,\ldots,0}]_{2^k}$ from $[(x+r)|_{k-2,\ldots,0}]_{2^k}$ at Step 14. If $\mathsf{needFPC} = 1$, i.e., $(x+r)|_{k-2,\ldots,0} \neq 2^{k-1}-1$, to verify whether the wrap-around occurs or not, the parties execute $\Pi_{\mathsf{FPC}}(k-1,\ \{[r|_\ell]_p\}_{\ell=0}^{k-2},\ (x+r)|_{k-2,\ldots,0})$ at Step 15. Then, the parties remove the shared mask $[r|_{k-2,\ldots,0}]_{2^k}$ from $[(x+r)|_{k-2,\ldots,0}]_{2^k}$ canceling the effect of the wrap-around at Step 16. After that, the parties get $[x|_{k-2,\ldots,0}]_{2^k}$. Next, they obtain $[\mathsf{msb}(x) \oplus \mathsf{msb}(r)]_2$ by masking and opening from Steps 17 to 23. Then, they remove the mask $\mathsf{msb}(r)$ and get $[\mathsf{msb}(x)]_2$ at Step 24.

How to Construct LT and EQ Protocols, i.e., Π_{LT} and Π_{EQ}. We can construct Π_{LT} and Π_{EQ} by replacing the MSB extraction protocol with ours in LT and EQ protocols of [22]. We assume $0 \leq a, b \leq 2^{k-1}-1$. In Π_{LT}, the parties compute the shares of the MSB of $[a-b]_{2^k} = [a]_{2^k} - [b]_{2^k}$ by Π_{msbExt} to run the LT operation. If a is smaller than b, $\mathsf{msb}(a-b)$ equals 1 and can be the output as the result of LT. If not, $\mathsf{msb}(a-b)$ equals 0 and can be the output. In Π_{EQ}, the parties invoke $\Pi_{\mathsf{LT}}([a]_{2^k}, [b]_{2^k})$ and $\Pi_{\mathsf{LT}}([b]_{2^k}, [a]_{2^k})$ in parallel. Note that $a = b$ holds if $(a < b) \oplus 1 = 1$ and $(b < a) \oplus 1 = 1$. Therefore, the parties compute the shares of the EQ result by $\mathsf{Mult}([(a<b) \oplus 1]_2,\ [(b<a) \oplus 1]_2)$.

4.7.3 Proposed Protocol of PDTE

Overview. We construct three protocols for each phase, i.e., feature selection $\Pi_{\mathsf{FSelection}}$ (Protocol 21), comparison Π_{Comp} and path evaluation Π_{PathEval} (Protocol 22). Then, we combine these protocols and construct the maliciously secure PDTE protocol with constant rounds and fairness, Π_{PDTE}.

To construct $\Pi_{\mathsf{FSelection}}$, we follow the typical algorithm of oblivious array read (e.g., the described algorithm in [72]), instead of the existing feature selection algorithm of [22] and instantiate the maliciously secure feature selection protocol with constant rounds and fairness by using our EQ protocol Π_{EQ}, $\mathsf{BitConv}$ and $\mathsf{DotProd}$ as building blocks. We note that the typical algorithm of oblivious array read in [72] is not constant-round protocol. Further, there is no EQ protocol achieving fairness and constant rounds except our EQ protocol Π_{EQ} although $\mathsf{BitConv}$ and $\mathsf{DotProd}$ with fairness and constant rounds have been already proposed.

To construct Π_{Comp} and Π_{PathEval}, we follow the algorithms of existing comparison and path evaluation protocols in [22] and instantiate the maliciously secure comparison and path evaluation protocols with constant rounds and fairness by using our LT, EQ and

shuffle protocols, i.e., Π_{LT}, Π_{EQ} and $\Pi_{\mathsf{TableShuffle}}$ and existing building blocks DotProd and OpenOne. We note that there are no LT, EQ or shuffle protocols achieving fairness and constant rounds except our LT, EQ and shuffle protocols, Π_{LT}, Π_{EQ} and $\Pi_{\mathsf{TableShuffle}}$ although DotProd and OpenOne with fairness and constant rounds have been already proposed.

Protocol 21 Feature Selection Protocol $\Pi_{\mathsf{FSelection}}$

Input: $[\mathsf{idx}]_{2^k}$, $\{[\mathsf{attr}_j]_{2^k}\}_{j=0}^{m-1}$ (s.t. $0 \leq \mathsf{idx} < m \leq 2^{k-1}-1$).
Output: $[\mathsf{attr}_{\mathsf{idx}}]_{2^k}$

1: **for** $j = 0, \ldots, m-1$ **do in parallel**
2: P_3 sets $[0]_{2^k,3} = (0,0,0)$.
3: $P_{i'}$ $(i' = 0,1,2)$ sets $[j]_{2^k,i'} = (j,0,0)$.
4: $[\mathsf{idx} == j]_2 \leftarrow \Pi_{\mathsf{EQ}}([\mathsf{idx}]_{2^k}, [j]_{2^k})$ // 18 rounds & $12k^2+22k+(138k-114)\log_2 p'+$ 7 bits in offline, 10 rounds & $36(k-1)\log_2 p' + 30k + 3$ bits in online
5: $[\mathsf{idx} == j]_{2^k} = \mathsf{BitConv}(2^k, [\mathsf{idx} == j]_2)$ // 2 rounds & $3k$ bits in offline, 1 round & $3k+1$ bits in online
6: **end for**
7: Return $[\mathsf{attr}_{\mathsf{idx}}]_{2^k} = \mathsf{DotProd}(([\mathsf{attr}_0]_{2^k}, \ldots, [\mathsf{attr}_{m-1}]_{2^k}), ([\mathsf{idx} == 0]_{2^k}, \ldots, [\mathsf{idx} == m-1]_{2^k}))$ // 1 round & $3k$ bits in offline, 1 round & $3k$ bits in online

Intuition of Protocol 21 (Feature Selection Phase). Protocol 21, $\Pi_{\mathsf{FSelection}}$, takes the shares of index $[\mathsf{idx}]_{2^k}$ (s.t. $\mathsf{idx} \in \mathbb{Z}_m$) and the array of shares $\{[\mathsf{attr}_j]_{2^k}\}_{j=0}^{m-1}$ and outputs $[\mathsf{attr}_{\mathsf{idx}}]_{2^k}$. In $\Pi_{\mathsf{FSelection}}$, the parties check whether $\mathsf{idx} == j$ obliviously by Π_{EQ} from Steps 1 to 4 and convert the output shares of Π_{EQ} over $\mathbb{Z}_2$ into the shares over $\mathbb{Z}_{2^k}$ by BitConv at Step 5 for $j = 0, \ldots, m-1$. Then, they choose $[\mathsf{attr}_{\mathsf{idx}}]_{2^k}$ obliviously by DotProd at Step 7.

How to Construct Comparison Protocol (Comparison Phase). Comparison protocol, Π_{Comp}, takes the shares of the attribute compared with the threshold values at each intermediate node $\{[\mathsf{attr}_{\mathsf{idx}_j}]_{2^k}\}_{j=0}^{2^h-2}$, the shares of the threshold values $\{[\mathsf{v}_j]_{2^k}\}_{j=0}^{2^h-2}$, and the shares of the conditional value that controls whether the LT or EQ is used as the comparison operation at each intermediate node $\{[\mathsf{cond}_j]_2\}_{j=0}^{2^h-2}$. It outputs the shares of comparison results $\{[\mathsf{comp}_j]_2\}_{j=0}^{2^h-2}$. In the same way as [22], the parties compute the results of LT and EQ in parallel by Π_{LT} and Π_{EQ} at Steps 2 and 3. Then, they choose either $[\mathsf{v}_j < \mathsf{attr}_{\mathsf{idx}_j}]_2$ or $[\mathsf{v}_j == \mathsf{attr}_{\mathsf{idx}_j}]_2$ as $[\mathsf{comp}_j]_2$ obliviously by DotProd, depending on $[\mathsf{cond}_j]_2$ for $j = 0, \ldots, 2^h-2$ in parallel.

Protocol 22 Path Evaluation Protocol Π_{PathEval}

Input: $\{[\mathsf{comp}_j]_2\}_{j=0}^{2^h-2}$, $\{[\mathsf{leafVal}_{j'}]_{2^k}\}_{j'=0}^{2^h-1}$, δ
Output: $[\mathsf{leafVal}_{j'}]_{2^k}$ where j' s.t. $\bigwedge_{\ell=0}^{h-1}(j'|_\ell == \mathsf{comp}_{\delta(j',\ell)}) = 1$.

1: **for** $j' = 0, \ldots, 2^h-1$ **do**
2: Initialize $\mathsf{Path}_{j'} = ([\mathsf{comp}_{\delta(j',0)}]_2, [\mathsf{comp}_{\delta(j',1)}]_2, \ldots, [\mathsf{comp}_{\delta(j',h-1)}]_2)$.
3: **for** $\ell = 0, \ldots, h-1$ **do**
4: $[c_{j',\ell}]_2 \leftarrow j'|_\ell \oplus [\mathsf{comp}_{\delta(j',\ell)}]_2 \oplus 1$ by picking up $[\mathsf{comp}_{\delta(j',\ell)}]_2$ from $\mathsf{Path}_{j'}$.
5: **end for**
6: Set $\mathcal{R}_{j'} = ([\mathsf{leafVal}_{j'}]_{2^k}, [c_{j',0}]_2, \ldots, [c_{j',h-1}]_2)$

7: **end for**

8: $\mathcal{R}'_0, \ldots, \mathcal{R}'_{2^h-1} \leftarrow \Pi_{\mathsf{TableShuffle}}(2^h, h+1, \mathcal{R}_0, \ldots, \mathcal{R}_{2^h-1})$ where $\mathcal{R}'_{j'} = ([\mathsf{leafVal}'_{j'}]_{2^k}, [c'_{j',0}]_2, \ldots, [c'_{j',h-1}]_2)$, $\mathsf{leafVal}'_{\pi(j')} = \mathsf{leafVal}_{j'}$ $(j' = 0, \ldots, 2^h - 1)$, $c'_{\pi(j'),\ell} = c_{j',\ell}$ $(j' = 0, \ldots, 2^h - 1;\ \ell = 0, \ldots, h-1)$ and a random permutation $\pi \in \mathcal{S}_{2^h}$ that no party knows.
// 4 rounds & $9 \cdot 2^h \cdot (h+1) \cdot (k+h)$ bits in offline, 3 rounds & $3 \cdot 2^h \cdot (h+1) \cdot (k+h)$ bits in online

9: Initialize $\mathsf{count}_{j'} = 0$ for $j' = 0, \ldots, 2^h - 1$.

10: **for** $j' = 0, \ldots, 2^h - 1; \ell = 0, \ldots, h-1$ **do in parallel**

11: Pick up $[c'_{j',\ell}]_2$ from $\mathcal{R}'_{j'}$. Then, P_i gets $c'_{j',\ell}$ by $\mathsf{OpenOne}(P_i, [c'_{j',\ell}]_2)$ for $i = 0, \ldots, 3$.
// 0 round & 0 bit in offline, 1 round & 4 bits in online

12: $\mathsf{count}_{j'} = \mathsf{count}_{j'} + 1$ if $c'_{j',\ell} = 1$.

13: **end for**

14: Return $[\mathsf{leafVal}'_{j'}]_{2^k}$ where $\mathsf{count}_{j'} = h$.

Intuition of Protocol 22 (Path Evaluation Phase). Protocol 22, Π_{PathEval}, takes the shares of the comparison result of intermediate nodes $\{[\mathsf{comp}_j]_2\}_{j=0}^{2^h-2}$, the shares of labels assigned to leaf nodes $\{[\mathsf{leafVal}_{j'}]_{2^k}\}_{j'=0}^{2^h-1}$, and mapping function δ. It outputs the shares of the label assigned to the leaf node of the correct path $[\mathsf{leafVal}_{j'}]_{2^k}$, where j' s.t. $\bigwedge_{\ell=0}^{h-1}(j'|_\ell == \mathsf{comp}_{\delta(j',\ell)}) = \bigwedge_{\ell=0}^{h-1}(j'|_\ell \oplus \mathsf{comp}_{\delta(j',\ell)} \oplus 1) = 1$. In the same way as [22], from Steps 1 to 4 of Π_{PathEval}, the parties check whether the comparison result $\mathsf{comp}_{\delta(j',\ell)}$ and the bit assigned to the branch of the path to the j'-th leaf node, $j'|_\ell$, match or not and outputs the shares of the matching result $[c_{j',\ell}]_2$. Then, the parties set the row vector of shares $\mathcal{R}_{j'}$ that includes the shares of j'-th leaf label $[\mathsf{leafVal}_{j'}]_{2^k}$ and the shares of the matching result bit $[c_{j',0}]_2, \ldots, [c_{j',h-1}]_2$ at Step 6. Next, the parties get the shuffled row vectors $\mathcal{R}'_0, \ldots, \mathcal{R}'_{2^h-1}$ by $\Pi_{\mathsf{TableShuffle}}(2^h, h+1, \mathcal{R}_0, \ldots, \mathcal{R}_{2^h-1})$ at Step 8. After that, by $\mathsf{OpenOne}$, the parties reconstruct the (shuffled) matching result $c'_{j',\ell}$ and increase the value of $\mathsf{count}_{j'}$ if $c'_{j',\ell} = 1$ from Steps 9 to 13. Finally, the parties output $[\mathsf{leafVal}'_{j'}]_{2^k}$, where $\mathsf{count}_{j'} = h$.

Note that $c'_{j',\ell}$ does not leak the positional information j'. An adversary can obtain no information about $\{\mathsf{comp}_{\delta(j',\ell)}\}_{\ell=0}^{2^h-2}$ or $\mathsf{leafVal}_{j'}$ from $c'_{j',\ell}$ thanks to the complete binary tree and $\Pi_{\mathsf{TableShuffle}}$. For example, we assume that $h = 2$. If the correct output leaf node is the leaf node $2(= 10_{(2)})$, it holds that $c_{0,0} = 1$, $c_{0,1} = 0$, $c_{1,0} = 0$, $c_{1,1} = 0$, $c_{2,0} = 1$, $c_{2,1} = 1$, $c_{3,0} = 0$, and $c_{3,1} = 1$. That is, an adversary gets all the 2-bit sequences ($00_{(2)}$, $01_{(2)}$, $10_{(2)}$, and $11_{(2)}$) from the shuffled matching result $c'_{j',\ell}$. As another example, if the correct output leaf node is the leaf node $3(= 11_{(2)})$, it holds that $c_{0,0} = 0$, $c_{0,1} = 0$, $c_{1,0} = 1$, $c_{1,1} = 0$, $c_{2,0} = 0$, $c_{2,1} = 1$, $c_{3,0} = 1$, and $c_{3,1} = 1$. An adversary also obtains all the 2-bit sequences ($00_{(2)}$, $01_{(2)}$, $10_{(2)}$, and $11_{(2)}$) from $c'_{j',\ell}$. Therefore, an adversary can obtain no information about j', $\{\mathsf{comp}_{\delta(j',\ell)}\}_{\ell=0}^{2^h-2}$ or $\mathsf{leafVal}_{j'}$ by reconstructing the shuffled matching result $c'_{j',\ell}$.

If the structure of tree is not a complete binary tree, reconstructing the shuffled matching result $c'_{j',\ell}$ may leak some information about j', $\{\mathsf{comp}_{\delta(j',\ell)}\}_{\ell=0}^{2^h-2}$ or $\mathsf{leafVal}_{j'}$. If the tree

Table 4.10: Communication complexity of the proposed protocols (Rounds: the number of communication rounds, Comm.: the number of (amortized) communication bits per all parties, L: modulus size, k: bit length of power-of-two ring, k': bit length for private compare, p': the smallest prime larger than k, h: height of tree, M: length of array, R: number of rows, C: number of columns, m: dimension of input attribute vector)

	Rounds		Comm.	
	Offline	Online	Offline	Online
$\Pi_{\mathsf{miniShuffle1}}$	1	0	$3M \log_2 L$	0
$\Pi_{\mathsf{miniShuffle2}}$	1	1	$2M \log_2 L$	$M \log_2 L$
$\Pi_{\mathsf{TableShuffle}}$	4	3	$9RC \log_2 L$	$3RC \log_2 L$
Π_{RSG}	0	0	0	0
Π_{FPC}	14	7	$(63k'+6)\log_2 p' + 9k + 2$	$18k' \log_2 p' + 3k$
Π_{msbExt}	17	9	$6k^2 + (69k-57)\log_2 p' + 11k + 2$	$15k + 18(k-1)\log_2 p'$
Π_{LT}	17	9	$6k^2 + (69k-57)\log_2 p' + 11k + 2$	$15k + 18(k-1)\log_2 p'$
Π_{EQ}	18	10	$12k^2 + (138k-114)\log_2 p' + 22k + 7$	$30k + 36(k-1)\log_2 p' + 3$
$\Pi_{\mathsf{FSelection}}$	21	12	$12k^2 m + (138k-114)m \log_2 p' + 25mk + 7m + 3k$	$36(k-1)m \log_2 p' + 33mk + 4m + 3k$
Π_{Comp}	19	11	$(2^h-1)\cdot(18k^2 + (207k-171)\log_2 p' + 11k + 22k + 12)$	$(2^h-1)\cdot(54(k-1)\log_2 p' + 45k + 6)$
Π_{PathEval}	4	4	$9 \cdot 2^h \cdot (h+1) \cdot (k+h)$	$3 \cdot 2^h \cdot (h+1) \cdot (k+h) + 4 \cdot 2^h \cdot h$
Π_{PDTE}	44	27	$(2^h-1)\cdot((12m+18)k^2 + (138km + 207k$ $-114m - 171)\log_2 p' + 25mk + 7m + 25k$ $+12) + 9 \cdot 2^h \cdot (h+1) \cdot (k+h)$	$(2^h-1)\cdot(36km - m54k - 54)\log_2 p'$ $+33mk + 4m + 48k + 6) + 3 \cdot 2^h$ $\cdot(h+1)\cdot(k+h) + 4 \cdot 2^h \cdot h$

is not a complete binary tree, the length of the path to a leaf node is different for each leaf node. For example, suppose that only one leaf node has a distance of 3 from the root node to itself, while the other leaf nodes have a distance of 2. If an adversary obtain the 3-bit sequence, $111_{(2)}$, then the adversary would know that the leaf node with a distance of 3 from the root node to itself is the correct leaf node. Therefore, the structure of tree must be a complete binary tree. That is, for all leaf nodes, the distance from the root node to the leaf node must be the same to hide some information about j', $\{\mathsf{comp}_{\delta(j',\ell)}\}_{\ell=0}^{2^h-2}$ or $\mathsf{leafVal}_{j'}$.

How to construct the PDTE protocol. Π_{PDTE} is our construction of PDTE that achieves malicious security with fairness and constant rounds. It takes the shares of input attributes $\{[\mathsf{attr}_i]_{2^k}\}_{i=0}^{m-1}$ and tree $\mathcal{T}$ and outputs the shares of the leaf on the correct path $[\mathsf{leafVal}_{j'}]_{2^k}$. It utilizes $\Pi_{\mathsf{FSelection}}$, Π_{Comp}, and Π_{PathEval} in each phase in the same way as [22], respectively.

4.7.4 Communication Complexities of Proposed Protocols with Fairness

Table 4.10 shows the communication complexities for the proposed protocols with fairness.

4.8 Security Proof of Proposed PDTE Protocols with Semi-honest Security

We follow the formal security definition of perfect security in the presence of one semi-honest corrupted party [5]. Loosely speaking, our schemes are composed of the UC secure building blocks and several operations without communications. Therefore, our scheme is secure as long as the building blocks are secure.

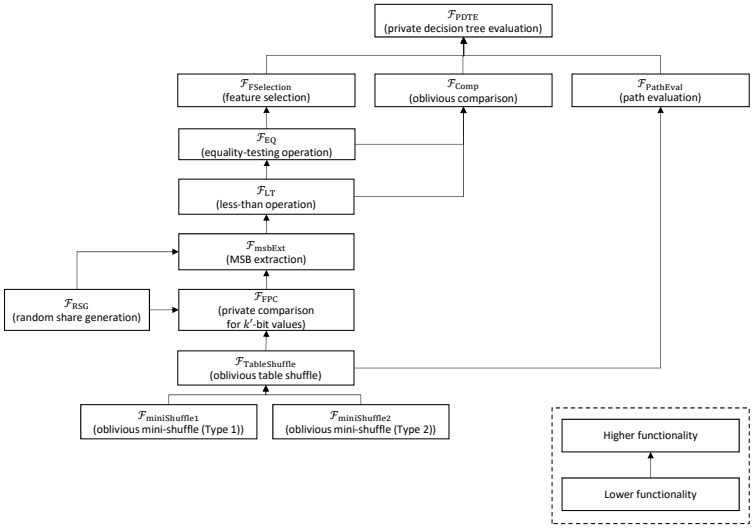

Figure 4.3: Dependency of ideal functionalities (Copyright(C)2022 IEICE, [1] Fig.2)

4.9 Security Proof of Proposed PDTE Protocols with Malicious Security and Fairness

We assume that the pseudo-random function F_L and fair 4PC building blocks are secure. We prove that our protocols compute the ideal functionalities securely with computational security and fairness in the presence of one malicious party.

Note that Kushilevitz et al. [95] showed that a protocol in which the security is proven with a black-box non-rewinding simulator assuming that the inputs of all parties are fixed before the execution (also known as input availability) achieves the universal composability [92]. That is, it is sufficient to prove the security of our protocols in the classical stand-alone setting by using the hybrid model if we assume that our protocols have the input availability in the same way as [5, 8].

Fig. 4.3 shows the dependency of ideal functionalities. Loosely speaking, we prove that the lower protocol computes the lower ideal functionality securely with computational security and fairness in the presence of one malicious party. Then, we prove that the higher protocol in the hybrid model of lower ideal functionality computes the higher ideal functionality securely with computational security and fairness in the presence of one malicious party. We show that the following theorems hold:

Theorem 1. *If F_L is a pseudo-random function and* CC *is secure, then the oblivious mini-shuffle protocol $\Pi_{\mathsf{miniShuffle1}}$ (Protocol 16) computes $\mathcal{F}_{\mathsf{miniShuffle1}}$ securely with computational security and fairness in the presence of one malicious party.*

Proof. If the adversary $\mathcal{A}$ corrupts one of the senders (i.e., P_0, P_1, or P_2), it is easy to

$\mathcal{F}_{\mathsf{miniShuffle1}}$ - (oblivious mini-shuffle (Type 1))

1. $P_{i'}$ sends the message $(\mathsf{miniShuffle1},\ \pi(\in \mathcal{S}_M),\ ([x_0]_{L,i'}, \ldots, [x_{M-1}]_{L,i'}))$ to $\mathcal{F}_{\mathsf{miniShuffle1}}$ for $i' = 0, 1, 2$.
2. P_3 sends the message $\mathsf{miniShuffle1}$.
3. After receiving the messages, $\mathcal{F}_{\mathsf{miniShuffle1}}$ reconstructs the values, $x_\ell \in \mathbb{Z}_L$ by using $[x_\ell]_{L,i'}$ (for $\ell = 0, \ldots, M-1;\ i' = 0, 1, 2$).
4. $\mathcal{F}_{\mathsf{miniShuffle1}}$ generates the random values $\lambda_{x'_{\pi(\ell)}}, \lambda_{x'_{\pi(\ell)},0}, \lambda_{x'_{\pi(\ell)},1}, \lambda_{x'_{\pi(\ell)},2} \in \mathbb{Z}_L$ (s.t. $\lambda_{x'_{\pi(\ell)}} = \lambda_{x'_{\pi(\ell)},0} + \lambda_{x'_{\pi(\ell)},1} + \lambda_{x'_{\pi(\ell)},2} \mod L$) and sets ${x'}_{\pi(\ell)} = x_\ell$ and $\mathsf{m}_{x'_{\pi(\ell)}} = x_\ell + \lambda_{x'_{\pi(\ell)}}$ for $\ell = 0, \ldots, M-1$.
5. $\mathcal{F}_{\mathsf{miniShuffle1}}$ sets each party's shares as follows:

$$
\begin{aligned}
[{x'}_{\pi(\ell)}]_{L,0} &= (\mathsf{m}_{x'_{\pi(\ell)}},\ \lambda_{x'_{\pi(\ell)},1},\ \lambda_{x'_{\pi(\ell)},2}) \\
[{x'}_{\pi(\ell)}]_{L,1} &= (\mathsf{m}_{x'_{\pi(\ell)}},\ \lambda_{x'_{\pi(\ell)},2},\ \lambda_{x'_{\pi(\ell)},0}) \\
[{x'}_{\pi(\ell)}]_{L,2} &= (\mathsf{m}_{x'_{\pi(\ell)}},\ \lambda_{x'_{\pi(\ell)},0},\ \lambda_{x'_{\pi(\ell)},1}) \\
[{x'}_{\pi(\ell)}]_{L,3} &= (\lambda_{x'_{\pi(\ell)},0}, \lambda_{x'_{\pi(\ell)},1}, \lambda_{x'_{\pi(\ell)},2})
\end{aligned}
$$

6. $\mathcal{F}_{\mathsf{miniShuffle1}}$ sends $([{x'}_0]_{L,i}, \ldots, [{x'}_{M-1}]_{L,i})$ to P_i for $i = 0, \ldots, 3$.

construct the simulator $\mathcal{S}$ and the (corrupted) party's view by using the party's inputs (and internal random coins) because the senders receive no messages from another party. Loosely speaking, $\mathcal{S}$ computes the view by using the party's inputs and F_L in the same way as in the actual protocol $\Pi_{\mathsf{miniShuffle1}}$ and sending the message $\mathsf{continue}$ to $\mathcal{F}_{\mathsf{miniShuffle1}}$. If $\mathcal{A}$ manipulates the sending message ${\lambda'}_{\pi(\ell),j}$ by applying the other permutation π' or adding the random value, $\mathcal{S}$ computes the party's view in the same way as $\mathcal{A}$'s cheating and sends the message abort to $\mathcal{F}_{\mathsf{miniShuffle1}}$.

Even if the adversary $\mathcal{A}$ corrupts the receiver P_3, we can construct $\mathcal{S}$ because P_3 just sets the received values as P_3's shares. For example, P_3 receives $[{x'}_{\pi(\ell)}]_{L,3} = (\lambda_{\ell,0}, \lambda_{\ell,1}, \lambda_{\ell,2})$. Then, we let the receiving messages sent from senders in P_3's view be ${\lambda'}^{(\overline{i'+1})}_{\pi(\ell),i'}$ sent by $P_{\overline{i'+1}}$ and ${\lambda'}^{(\overline{i'-1})}_{\pi(\ell),i'}$ sent by $P_{\overline{i'-1}}$ for $i' = 0, 1, 2$. $\mathcal{S}$ sets ${\lambda'}^{(\overline{i'+1})}_{\pi(\ell),i'} = \lambda_{\ell,i'}$ and ${\lambda'}^{(\overline{i'-1})}_{\pi(\ell),i'} = \mathcal{H}(\lambda_{\ell,i'})$.

Therefore, we can construct $\mathcal{S}$ for every corrupted party. $\Pi_{\mathsf{miniShuffle1}}$ satisfies Definition 1. Hence, Theorem 1 holds. □

Theorem 2. *If F_L is a pseudo-random function and CC is secure, the oblivious mini-shuffle protocol $\Pi_{\mathsf{miniShuffle2}}$ computes $\mathcal{F}_{\mathsf{miniShuffle2}}$ securely with computational security and fairness in the presence of one malicious party.*

$\mathcal{F}_{\mathsf{miniShuffle2}}$ - (oblivious mini-shuffle (Type 2))

1. P_3 sends the message $(\mathsf{miniShuffle2},\ \pi(\in \mathcal{S}_M),\ ([x_0]_{L,3}, \ldots, [x_{M-1}]_{L,3}))$ to $\mathcal{F}_{\mathsf{miniShuffle2}}$.
2. $P_{\overline{i'-1}}$ sends the message $(\mathsf{miniShuffle2},\ \pi,\ ([x_0]_{L,\overline{i'-1}}, \ldots, [x_{M-1}]_{L,\overline{i'-1}}))$ to $\mathcal{F}_{\mathsf{miniShuffle2}}$ where $i' \in \{0,1,2\}$.
3. $P_{\overline{i'+1}}$ sends the message $(\mathsf{miniShuffle2},\ \pi,\ ([x_0]_{L,\overline{i'+1}}, \ldots, [x_{M-1}]_{L,\overline{i'+1}}))$ to $\mathcal{F}_{\mathsf{miniShuffle2}}$ where $i' \in \{0,1,2\}$.
4. $P_{i'}$ sends the message $\mathsf{miniShuffle2}$ where $i' \in \{0,1,2\}$.
5. After receiving the messages, $\mathcal{F}_{\mathsf{miniShuffle2}}$ reconstructs the values, $x_\ell \in \mathbb{Z}_L$ by using $[x_\ell]_{L,3}$, $[x_\ell]_{L,\overline{i'-1}}$, and $[x_\ell]_{L,\overline{i'+1}}$ (for $\ell = 0, \ldots, M-1$).
6. $\mathcal{F}_{\mathsf{miniShuffle2}}$ generates the random values $\lambda_{x'_{\pi(\ell)}}, \lambda_{x'_{\pi(\ell)},0}, \lambda_{x'_{\pi(\ell)},1}, \lambda_{x'_{\pi(\ell)},2} \in \mathbb{Z}_L$ (s.t. $\lambda_{x'_{\pi(\ell)}} = \lambda_{x'_{\pi(\ell)},0} + \lambda_{x'_{\pi(\ell)},1} + \lambda_{x'_{\pi(\ell)},2} \mod L$) and sets $x'_{\pi(\ell)} = x_\ell$ and $\mathsf{m}_{x'_{\pi(\ell)}} = x_\ell + \lambda_{x'_{\pi(\ell)}}$ for $\ell = 0, \ldots, M-1$.
7. $\mathcal{F}_{\mathsf{miniShuffle2}}$ sets each party's shares as follows:

$$
\begin{aligned}
[x'_{\pi(\ell)}]_{L,0} &= (\mathsf{m}_{x'_{\pi(\ell)}},\ \lambda_{x'_{\pi(\ell)},1},\ \lambda_{x'_{\pi(\ell)},2}) \\
[x'_{\pi(\ell)}]_{L,1} &= (\mathsf{m}_{x'_{\pi(\ell)}},\ \lambda_{x'_{\pi(\ell)},2},\ \lambda_{x'_{\pi(\ell)},0}) \\
[x'_{\pi(\ell)}]_{L,2} &= (\mathsf{m}_{x'_{\pi(\ell)}},\ \lambda_{x'_{\pi(\ell)},0},\ \lambda_{x'_{\pi(\ell)},1}) \\
[x'_{\pi(\ell)}]_{L,3} &= (\lambda_{x'_{\pi(\ell)},0}, \lambda_{x'_{\pi(\ell)},1}, \lambda_{x'_{\pi(\ell)},2})
\end{aligned}
$$

8. $\mathcal{F}_{\mathsf{miniShuffle2}}$ sends $([x'_0]_{L,i}, \ldots, [x'_{M-1}]_{L,i})$ to P_i for $i = 0, \ldots, 3$.

Proof. We can prove that Theorem 2 holds in the same way as the proof of Theorem 1. □

Theorem 3. *The oblivious shuffle protocol for table data $\Pi_{\mathsf{TableShuffle}}$ (Protocol 17) in the $(\mathcal{F}_{\mathsf{miniShuffle1}}, \mathcal{F}_{\mathsf{miniShuffle2}})$-hybrid model computes $\mathcal{F}_{\mathsf{TableShuffle}}$ securely with computational security and fairness in the presence of one malicious party.*

Proof. We replace calling $\Pi_{\mathsf{miniShuffle1}}$ and $\Pi_{\mathsf{miniShuffle2}}$ with invoking the ideal functionalities $\mathcal{F}_{\mathsf{miniShuffle1}}$ and $\mathcal{F}_{\mathsf{miniShuffle2}}$ and prove the security of $\Pi^{(\mathcal{F}_{\mathsf{miniShuffle1}}, \mathcal{F}_{\mathsf{miniShuffle2}})}_{\mathsf{TableShuffle}}$. $\Pi^{(\mathcal{F}_{\mathsf{miniShuffle1}}, \mathcal{F}_{\mathsf{miniShuffle2}})}_{\mathsf{TableShuffle}}$ consists of invoking the ideal functionalities $\mathcal{F}_{\mathsf{miniShuffle1}}$ and $\mathcal{F}_{\mathsf{miniShuffle2}}$ and computations without communications. Therefore, the simulator $\mathcal{S}$ can be composed in the case where a party is corrupted. Hence, $\Pi^{(\mathcal{F}_{\mathsf{miniShuffle1}}, \mathcal{F}_{\mathsf{miniShuffle2}})}_{\mathsf{TableShuffle}}$ satisfies Definition 1. Therefore, Theorem 3 holds. □

$\mathcal{F}_{\mathsf{TableShuffle}}$ - (oblivious table shuffle)

1. P_i sends the message $(\mathsf{shuffle},\ R,\ C,\ [\vec{x}_0]_i = ([x_{0,0}]_{L_0,i}, [x_{0,1}]_{L_1,i}, \ldots, [x_{0,C-1}]_{L_{C-1},i}), \ldots, [\vec{x}_{R-1}]_i = ([x_{R-1,0}]_{L_0,i}, [x_{R-1,1}]_{L_1,i}, \ldots, [x_{R-1,C-1}]_{L_{C-1},i})$ to $\mathcal{F}_{\mathsf{TableShuffle}}$ for $i = 0, \ldots, 3$.

2. After receiving the messages, $\mathcal{F}_{\mathsf{TableShuffle}}$ reconstructs the values, $x_{\ell',\ell} \in \mathbb{Z}_{L_\ell}$ by using the received shares for $\ell' = 0, \ldots, R-1;\ \ell = 0, \ldots, C-1$.

3. $\mathcal{F}_{\mathsf{TableShuffle}}$ generates the random permutation $\pi \in S_R$.

4. $\mathcal{F}_{\mathsf{TableShuffle}}$ generates the random values $\lambda_{x'_{\pi(\ell'),\ell}}, \lambda_{x'_{\pi(\ell'),\ell},0}, \lambda_{x'_{\pi(\ell'),\ell},1}, \lambda_{x'_{\pi(\ell'),\ell},2} \in \mathbb{Z}_L$ (s.t. $\lambda_{x'_{\pi(\ell'),\ell}} = \lambda_{x'_{\pi(\ell'),\ell},0} + \lambda_{x'_{\pi(\ell'),\ell},1} + \lambda_{x'_{\pi(\ell'),\ell},2} \mod L$) and sets $x'_{\pi(\ell'),\ell} = x_{\ell',\ell}$ and $\mathsf{m}_{x'_{\pi(\ell'),\ell}} = x_{\ell',\ell} + \lambda_{x'_{\pi(\ell'),\ell}}$ for $\ell' = 0, \ldots, R-1;\ \ell = 0, \ldots, C-1$.

5. $\mathcal{F}_{\mathsf{TableShuffle}}$ sets each party's shares as follows:

$$
\begin{aligned}
[x'_{\pi(\ell'),\ell}]_{L,0} &= (\mathsf{m}_{x'_{\pi(\ell'),\ell}},\ \lambda_{x'_{\pi(\ell'),\ell},1},\ \lambda_{x'_{\pi(\ell'),\ell},2}) \\
[x'_{\pi(\ell'),\ell}]_{L,1} &= (\mathsf{m}_{x'_{\pi(\ell'),\ell}},\ \lambda_{x'_{\pi(\ell'),\ell},2},\ \lambda_{x'_{\pi(\ell'),\ell},0}) \\
[x'_{\pi(\ell'),\ell}]_{L,2} &= (\mathsf{m}_{x'_{\pi(\ell'),\ell}},\ \lambda_{x'_{\pi(\ell'),\ell},0},\ \lambda_{x'_{\pi(\ell'),\ell},1}) \\
[x'_{\pi(\ell'),\ell}]_{L,3} &= (\lambda_{x'_{\pi(\ell'),\ell},0}, \lambda_{x'_{\pi(\ell'),\ell},1}, \lambda_{x'_{\pi(\ell'),\ell},2})
\end{aligned}
$$

6. $\mathcal{F}_{\mathsf{TableShuffle}}$ sends $[\vec{x'}_0]_i = ([x'_{0,0}]_{L_0,i}, [x'_{0,1}]_{L_1,i}, \ldots, [x'_{0,C-2}]_{L_{C-2},i}, [x'_{0,C-1}]_{L_{C-1},i}), \ldots, [\vec{x'}_{R-1}]_i = ([x'_{R-1,0}]_{L_0,i}, [x'_{R-1,1}]_{L_1,i}, \ldots, [x'_{R-1,C-2}]_{L_{C-2},i}, [x'_{R-1,C-1}]_{L_{C-1},i})$ to P_i for $i = 0, \ldots, 3$.

$\mathcal{F}_{\mathsf{RSG}}$ - (random share generation)

1. Parties send the message $(\mathsf{RSG},\ L)$ to $\mathcal{F}_{\mathsf{RSG}}$.

2. After receiving the message, $\mathcal{F}_{\mathsf{RSG}}$ generates the random value $\mathsf{m}_r, \lambda_{r,j'} \in \mathbb{Z}_L$ for $j' = 0, \ldots, 2$.

3. $\mathcal{F}_{\mathsf{RSG}}$ sends $[r]_{L,j'} = (\mathsf{m}_r, \lambda_{r,\overline{j'+1}}, \lambda_{r,\overline{j'-1}})$ to $P_{j'}$ for $j' = 0, \ldots, 2$ and $[r]_{L,3} = (\lambda_{r,0}, \lambda_{r,1}, \lambda_{r,2})$ to P_3. We note that $r = \mathsf{m}_r - (\sum_{j'=0}^{2} \lambda_{r,j'}) \mod L$.

$\mathcal{F}_{\mathsf{FPC}}$ - (private comparison for k'-bit values)

1. Each party P_i sends the message $(\mathsf{FPC},\ k', \{[x|_\ell]_{p,i}\}_{\ell=0}^{k'-1}, r)$ (s.t. $r \in \{0,1\}^{k'}$ and $x|_\ell \in \mathbb{Z}_2$) to $\mathcal{F}_{\mathsf{FPC}}$ for $i = 0, \ldots, 3$.
2. After receiving the messages, $\mathcal{F}_{\mathsf{FPC}}$ reconstructs $x|_\ell$ by using $[x|_\ell]_{p,i}$ (for $i = 0, \ldots, 3;\ \ell = 0, \ldots, k'-1$). Then, $\mathcal{F}_{\mathsf{FPC}}$ computes $x = \sum_{\ell=0}^{k'-1} 2^\ell \cdot x|_\ell$.
3. $\mathcal{F}_{\mathsf{FPC}}$ computes the comparison result bit, $b = (x > r)$.
4. $\mathcal{F}_{\mathsf{FPC}}$ generates the random values $\lambda_b, \lambda_{b,0}, \lambda_{b,1}, \lambda_{b,2} \in \mathbb{Z}_{2^k}$ (s.t. $\lambda_b = \lambda_{b,0} + \lambda_{b,1} + \lambda_{b,2} \mod 2^k$) and sets $\mathsf{m}_b = b + \lambda_b \mod 2^k$.
5. $\mathcal{F}_{\mathsf{FPC}}$ sets each party's shares as follows:

$$
\begin{aligned}
[b]_{2^k,0} &= (\mathsf{m}_b,\ \lambda_{b,1},\ \lambda_{b,2}) \\
[b]_{2^k,1} &= (\mathsf{m}_b,\ \lambda_{b,2},\ \lambda_{b,0}) \\
[b]_{2^k,2} &= (\mathsf{m}_b,\ \lambda_{b,0},\ \lambda_{b,1}) \\
[b]_{2^k,3} &= (\lambda_{b,0},\ \lambda_{b,1},\ \lambda_{b,2})
\end{aligned}
$$

6. $\mathcal{F}_{\mathsf{FPC}}$ sends $[b]_{2^k,i}$ to P_i for $i = 0, \ldots, 3$.

$\mathcal{F}_{\mathsf{msbExt}}$ - (MSB extraction)

1. Each party P_i sends the message $(\mathsf{msbExt},\ [x]_{2^k,i})$ to $\mathcal{F}_{\mathsf{msbExt}}$ for $i = 0, \ldots, 3$.
2. After receiving the messages, $\mathcal{F}_{\mathsf{msbExt}}$ reconstructs the values, x by using the received messages. Then, $\mathcal{F}_{\mathsf{msbExt}}$ computes $\mathsf{msb}(x) = x|_{k-1} \in \mathbb{Z}_2$.
3. $\mathcal{F}_{\mathsf{msbExt}}$ generates the random values $\lambda_{\mathsf{msb}(x)}, \lambda_{\mathsf{msb}(x),0}, \lambda_{\mathsf{msb}(x),1}, \lambda_{\mathsf{msb}(x),2} \in \mathbb{Z}_2$ (s.t. $\lambda_{\mathsf{msb}(x)} = \lambda_{\mathsf{msb}(x),0} \oplus \lambda_{\mathsf{msb}(x),1} \oplus \lambda_{\mathsf{msb}(x),2} \mod 2$) and sets $\mathsf{m}_{\mathsf{msb}(x)} = \mathsf{msb}(x) \oplus \lambda_{\mathsf{msb}(x)} \mod 2$.
4. $\mathcal{F}_{\mathsf{msbExt}}$ sets each party's shares as follows:

$$
\begin{aligned}
[\mathsf{msb}(x)]_{2,0} &= (\mathsf{m}_{\mathsf{msb}(x)},\ \lambda_{\mathsf{msb}(x),1},\ \lambda_{\mathsf{msb}(x),2}) \\
[\mathsf{msb}(x)]_{2,1} &= (\mathsf{m}_{\mathsf{msb}(x)},\ \lambda_{\mathsf{msb}(x),2},\ \lambda_{\mathsf{msb}(x),0}) \\
[\mathsf{msb}(x)]_{2,2} &= (\mathsf{m}_{\mathsf{msb}(x)},\ \lambda_{\mathsf{msb}(x),0},\ \lambda_{\mathsf{msb}(x),1}) \\
[\mathsf{msb}(x)]_{2,3} &= (\lambda_{\mathsf{msb}(x),0},\ \lambda_{\mathsf{msb}(x),1},\ \lambda_{\mathsf{msb}(x),2})
\end{aligned}
$$

5. $\mathcal{F}_{\mathsf{msbExt}}$ sends $[\mathsf{msb}(x)]_{2,i}$ to P_i for $i = 0, \ldots, 3$.

Theorem 4. *If the pseudo-random function F_L is secure, then the RSG protocol Π_{RSG} (Protocol 18) computes $\mathcal{F}_{\mathsf{RSG}}$ securely with computational security and fairness in the presence of one malicious party.*

Proof. Π_{RSG} consists of invoking the pseudo-random function F_L without communications. Therefore, the simulator $\mathcal{S}$ can be composed in the case where a party is corrupted. Hence, Π_{RSG} satisfies Definition 1. Therefore, Theorem 4 holds. □

Theorem 5. *If the pseudo-random function F_L and fair 4PC building blocks (i.e.,* Mult*,* BitConv*,* DotProd*, and* OpenOne*) are secure, then the fair and PC protocol for k'-bit values Π_{FPC} (Protocol 19) in the $(\mathcal{F}_{\mathsf{TableShuffle}}, \mathcal{F}_{\mathsf{RSG}})$-hybrid model computes $\mathcal{F}_{\mathsf{FPC}}$ securely with computational security and fairness in the presence of one malicious party.*

Proof. We replace calling $\Pi_{\mathsf{TableShuffle}}$ and Π_{RSG} with invoking the ideal functionality $\mathcal{F}_{\mathsf{TableShuffle}}$ and $\mathcal{F}_{\mathsf{RSG}}$ and prove the security of $\Pi_{\mathsf{FPC}}^{\mathcal{F}_{\mathsf{TableShuffle}},\mathcal{F}_{\mathsf{RSG}}}$. $\Pi_{\mathsf{FPC}}^{\mathcal{F}_{\mathsf{TableShuffle}},\mathcal{F}_{\mathsf{RSG}}}$ consists of invoking the ideal functionalities $\mathcal{F}_{\mathsf{TableShuffle}}$ and $\mathcal{F}_{\mathsf{RSG}}$, the fair 4PC building blocks, and computations without communications. Therefore, the simulator $\mathcal{S}$ can be composed in the case where a party is corrupted. Hence, $\Pi_{\mathsf{FPC}}^{\mathcal{F}_{\mathsf{TableShuffle}},\mathcal{F}_{\mathsf{RSG}}}$ satisfies Definition 1. Therefore, Theorem 5 holds. □

Theorem 6. *If the pseudo-random function F_L and fair 4PC building blocks are secure, then the MSB extraction protocol Π_{msbExt} (Protocol 20) in the $(\mathcal{F}_{\mathsf{FPC}}, \mathcal{F}_{\mathsf{RSG}})$-hybrid model computes $\mathcal{F}_{\mathsf{msbExt}}$ securely with computational security and fairness in the presence of one malicious party.*

Proof. We replace calling Π_{FPC} and Π_{RSG} with invoking the ideal functionality $\mathcal{F}_{\mathsf{FPC}}$ and $\mathcal{F}_{\mathsf{RSG}}$ and prove the security of $\Pi_{\mathsf{msbExt}}^{\mathcal{F}_{\mathsf{FPC}},\mathcal{F}_{\mathsf{RSG}}}$. $\Pi_{\mathsf{msbExt}}^{\mathcal{F}_{\mathsf{FPC}},\mathcal{F}_{\mathsf{RSG}}}$ consists of invoking the ideal functionalities $\mathcal{F}_{\mathsf{FPC}}$ and $\mathcal{F}_{\mathsf{RSG}}$, the fair 4PC building blocks and computations without communications. Therefore, the simulator $\mathcal{S}$ can be composed in the case where a party is corrupted. Hence, $\Pi_{\mathsf{msbExt}}^{\mathcal{F}_{\mathsf{FPC}},\mathcal{F}_{\mathsf{RSG}}}$ satisfies Definition 1. Therefore, Theorem 6 holds. □

Theorem 7. *The less-than protocol Π_{LT} in the $\mathcal{F}_{\mathsf{msbExt}}$-hybrid model computes $\mathcal{F}_{\mathsf{LT}}$ securely with computational security and fairness in the presence of one malicious party.*

Proof. We replace calling Π_{msbExt} with invoking the ideal functionality $\mathcal{F}_{\mathsf{msbExt}}$ and prove the security of $\Pi_{\mathsf{LT}}^{\mathcal{F}_{\mathsf{msbExt}}}$. $\Pi_{\mathsf{LT}}^{\mathcal{F}_{\mathsf{msbExt}}}$ consists of invoking the ideal functionality $\mathcal{F}_{\mathsf{msbExt}}$ and computations without communications. Therefore, the simulator $\mathcal{S}$ can be composed in the case where a party is corrupted. Hence, $\Pi_{\mathsf{LT}}^{\mathcal{F}_{\mathsf{msbExt}}}$ satisfies Definition 1. Therefore, Theorem 7 holds. □

Theorem 8. *If* Mult *is secure, then the equality-testing protocol Π_{EQ} in the $\mathcal{F}_{\mathsf{LT}}$-hybrid model computes $\mathcal{F}_{\mathsf{EQ}}$ securely with computational security and fairness in the presence of one malicious party.*

Proof. We replace calling Π_{LT} with invoking the ideal functionality $\mathcal{F}_{\mathsf{LT}}$ and prove the security of $\Pi_{\mathsf{EQ}}^{\mathcal{F}_{\mathsf{LT}}}$. $\Pi_{\mathsf{EQ}}^{\mathcal{F}_{\mathsf{LT}}}$ consists of invoking the ideal functionality $\mathcal{F}_{\mathsf{LT}}$, executing Mult, and computations without communications. Therefore, the simulator $\mathcal{S}$ can be composed in the

$\mathcal{F}_{\mathsf{LT}}$ - (less-than operation)

1. Each party P_i sends the message $(\mathsf{LT},\ [a]_{2^k,i},\ [b]_{2^k,i})$ to $\mathcal{F}_{\mathsf{LT}}$ for $i = 0, \ldots, 3$.
2. After receiving the messages, $\mathcal{F}_{\mathsf{LT}}$ reconstructs a and b by using the received messages. Then, $\mathcal{F}_{\mathsf{LT}}$ computes the comparison result bit $c = (a < b)$.
3. $\mathcal{F}_{\mathsf{LT}}$ generates the random values $\lambda_c, \lambda_{c,0}, \lambda_{c,1}, \lambda_{c,2} \in \mathbb{Z}_2$ (s.t. $\lambda_c = \lambda_{c,0} \oplus \lambda_{c,1} \oplus \lambda_{c,2} \mod 2$) and sets $\mathsf{m}_c = c \oplus \lambda_c \mod 2$.
4. $\mathcal{F}_{\mathsf{LT}}$ sets each party's shares as follows:

$$
\begin{aligned}
[c]_{2,0} &= (\mathsf{m}_c,\ \lambda_{c,1},\ \lambda_{c,2}) \\
[c]_{2,1} &= (\mathsf{m}_c,\ \lambda_{c,2},\ \lambda_{c,0}) \\
[c]_{2,2} &= (\mathsf{m}_c,\ \lambda_{c,0},\ \lambda_{c,1}) \\
[c]_{2,3} &= (\lambda_{c,0},\ \lambda_{c,1},\ \lambda_{c,2})
\end{aligned}
$$

5. $\mathcal{F}_{\mathsf{LT}}$ sends $[c]_{2,i}$ to P_i for $i = 0, \ldots, 3$.

$\mathcal{F}_{\mathsf{EQ}}$ - (equality-testing operation)

1. Each party P_i sends the message $(\mathsf{EQ},\ [a]_{2^k,i},\ [b]_{2^k,i})$ to $\mathcal{F}_{\mathsf{EQ}}$ for $i = 0, \ldots, 3$.
2. After receiving the messages, $\mathcal{F}_{\mathsf{EQ}}$ reconstructs a and b by using the received messages. Then, $\mathcal{F}_{\mathsf{EQ}}$ computes the comparison result bit $c = (a == b)$.
3. $\mathcal{F}_{\mathsf{EQ}}$ generates the random values $\lambda_c, \lambda_{c,0}, \lambda_{c,1}, \lambda_{c,2} \in \mathbb{Z}_2$ (s.t. $\lambda_c = \lambda_{c,0} \oplus \lambda_{c,1} \oplus \lambda_{c,2} \mod 2$) and sets $\mathsf{m}_c = c \oplus \lambda_c \mod 2$.
4. $\mathcal{F}_{\mathsf{EQ}}$ sets each party's shares as follows:

$$
\begin{aligned}
[c]_{2,0} &= (\mathsf{m}_c,\ \lambda_{c,1},\ \lambda_{c,2}) \\
[c]_{2,1} &= (\mathsf{m}_c,\ \lambda_{c,2},\ \lambda_{c,0}) \\
[c]_{2,2} &= (\mathsf{m}_c,\ \lambda_{c,0},\ \lambda_{c,1}) \\
[c]_{2,3} &= (\lambda_{c,0},\ \lambda_{c,1},\ \lambda_{c,2})
\end{aligned}
$$

5. $\mathcal{F}_{\mathsf{EQ}}$ sends $[c]_{2,i}$ to P_i for $i = 0, \ldots, 3$.

case where a party is corrupted. Hence, $\Pi_{\mathsf{EQ}}^{\mathcal{F}_{\mathsf{LT}}}$ satisfies Definition 1. Therefore, Theorem 8 holds. □

$\mathcal{F}_{\mathsf{FSelection}}$ - (feature selection)

1. Each party P_i sends the message $(\mathsf{FSelection}, [\mathsf{idx}]_{2^k,i}, [\mathsf{attr}_0]_{2^k,i}, \ldots, [\mathsf{attr}_{m-1}]_{2^k,i})$ to $\mathcal{F}_{\mathsf{FSelection}}$ for $i = 0, \ldots, 3$.

2. After receiving the messages, $\mathcal{F}_{\mathsf{FSelection}}$ reconstructs idx and $\{\mathsf{attr}_j\}_{j=0}^{m-1}$ by using the received messages where $0 \leq \mathsf{idx} < m \leq 2^{k-1} - 1$ and $0 \leq \mathsf{attr}_j \leq 2^{k-1} - 1$. Then, $\mathcal{F}_{\mathsf{FSelection}}$ chooses the idx-th attribute, $\mathsf{attr}_{\mathsf{idx}}$.

3. $\mathcal{F}_{\mathsf{FSelection}}$ generates the random values $\lambda_{\mathsf{attr}_{\mathsf{idx}}}, \lambda_{\mathsf{attr}_{\mathsf{idx}},0}, \lambda_{\mathsf{attr}_{\mathsf{idx}},1}, \lambda_{\mathsf{attr}_{\mathsf{idx}},2} \in \mathbb{Z}_{2^k}$ (s.t. $\lambda_{\mathsf{attr}_{\mathsf{idx}}} = \lambda_{\mathsf{attr}_{\mathsf{idx}},0} + \lambda_{\mathsf{attr}_{\mathsf{idx}},1} + \lambda_{\mathsf{attr}_{\mathsf{idx}},2} \mod 2^k$) and sets $\mathsf{m}_{\mathsf{attr}_{\mathsf{idx}}} = \mathsf{attr}_{\mathsf{idx}} + \lambda_{\mathsf{attr}_{\mathsf{idx}}} \mod 2^k$.

4. $\mathcal{F}_{\mathsf{FSelection}}$ sets each party's shares as follows:

$$
\begin{aligned}
[\mathsf{attr}_{\mathsf{idx}}]_{2^k,0} &= (\mathsf{m}_{\mathsf{attr}_{\mathsf{idx}}},\ \lambda_{\mathsf{attr}_{\mathsf{idx}},1},\ \lambda_{\mathsf{attr}_{\mathsf{idx}},2}) \\
[\mathsf{attr}_{\mathsf{idx}}]_{2^k,1} &= (\mathsf{m}_{\mathsf{attr}_{\mathsf{idx}}},\ \lambda_{\mathsf{attr}_{\mathsf{idx}},2},\ \lambda_{\mathsf{attr}_{\mathsf{idx}},0}) \\
[\mathsf{attr}_{\mathsf{idx}}]_{2^k,2} &= (\mathsf{m}_{\mathsf{attr}_{\mathsf{idx}}},\ \lambda_{\mathsf{attr}_{\mathsf{idx}},0},\ \lambda_{\mathsf{attr}_{\mathsf{idx}},1}) \\
[\mathsf{attr}_{\mathsf{idx}}]_{2^k,3} &= (\lambda_{\mathsf{attr}_{\mathsf{idx}},0},\ \lambda_{\mathsf{attr}_{\mathsf{idx}},1},\ \lambda_{\mathsf{attr}_{\mathsf{idx}},2})
\end{aligned}
$$

5. $\mathcal{F}_{\mathsf{FSelection}}$ sends $[\mathsf{attr}_{\mathsf{idx}}]_{2^k,i}$ to P_i for $i = 0, \ldots, 3$.

Theorem 9. *If* DotProd *and* BitConv *are secure, then the feature selection protocol* $\Pi_{\mathsf{FSelection}}$ *(Protocol 21) in the* $\mathcal{F}_{\mathsf{EQ}}$*-hybrid model computes* $\mathcal{F}_{\mathsf{FSelection}}$ *securely with computational security and fairness in the presence of one malicious party.*

Proof. We replace calling Π_{EQ} with invoking the ideal functionality $\mathcal{F}_{\mathsf{EQ}}$ and prove the security of $\Pi_{\mathsf{FSelection}}^{\mathcal{F}_{\mathsf{EQ}}}$. $\Pi_{\mathsf{FSelection}}^{\mathcal{F}_{\mathsf{EQ}}}$ consists of invoking the ideal functionality $\mathcal{F}_{\mathsf{EQ}}$, executing BitConv and DotProd, and computations without communications. Therefore, the simulator $\mathcal{S}$ can be composed in the case where a party is corrupted. Hence, $\Pi_{\mathsf{FSelection}}^{\mathcal{F}_{\mathsf{EQ}}}$ satisfies Definition 1. Therefore, Theorem 9 holds. □

Theorem 10. *If* DotProd *is secure, then the comparison protocol* Π_{Comp} *in the* $(\mathcal{F}_{\mathsf{LT}}, \mathcal{F}_{\mathsf{EQ}})$*-hybrid model computes* $\mathcal{F}_{\mathsf{Comp}}$ *securely with computational security and fairness in the presence of one malicious party.*

Proof. We replace calling Π_{LT} and Π_{EQ} with invoking the ideal functionalities $\mathcal{F}_{\mathsf{LT}}$ and Π_{EQ}, respectively. We prove the security of $\Pi_{\mathsf{Comp}}^{\mathcal{F}_{\mathsf{LT}},\mathcal{F}_{\mathsf{EQ}}}$. $\Pi_{\mathsf{Comp}}^{\mathcal{F}_{\mathsf{LT}},\mathcal{F}_{\mathsf{EQ}}}$ consists of invoking the ideal functionalities $\mathcal{F}_{\mathsf{LT}}$ and $\mathcal{F}_{\mathsf{EQ}}$, executing DotProd, and computations without communications. Therefore, the simulator $\mathcal{S}$ can be composed in the case where a party is corrupted. Hence, $\Pi_{\mathsf{Comp}}^{\mathcal{F}_{\mathsf{LT}},\mathcal{F}_{\mathsf{EQ}}}$ satisfies Definition 1. Therefore, Theorem 10 holds. □

$\mathcal{F}_{\mathsf{Comp}}$ - (oblivious comparison)

1. Each party P_i sends the message $(\mathsf{Comp}, \{[\mathsf{attr}_{\mathsf{idx}}]_{2^k,i}\}_{j=0}^{2^h-2}, \{[\mathsf{v}_j]_{2^k,i}\}_{j=0}^{2^h-2}, \{[\mathsf{cond}_j]_{2,i}\}_{j=0}^{2^h-2})$ to $\mathcal{F}_{\mathsf{Comp}}$ for $i = 0, \ldots, 3$.

2. After receiving the messages, $\mathcal{F}_{\mathsf{Comp}}$ reconstructs $\{\mathsf{attr}_{\mathsf{idx}_j}\}_{j=0}^{2^h-2}$, $\{\mathsf{v}_j\}_{j=0}^{2^h-2}$ and $\{\mathsf{cond}_j\}_{j=0}^{2^h-2}$ by using the received messages. Then, $\mathcal{F}_{\mathsf{Comp}}$ sets the comparison result bit comp_j for $j = 0, \ldots, 2^h-2$. If $\mathsf{cond}_j = 1$, $\mathsf{comp} = (\mathsf{v}_j < \mathsf{attr}_{\mathsf{idx}_j})$. If $\mathsf{cond}_j = 0$, $\mathsf{comp} = (\mathsf{v}_j == \mathsf{attr}_{\mathsf{idx}_j})$.

3. $\mathcal{F}_{\mathsf{Comp}}$ generates the random values $\lambda_{\mathsf{comp}_j}, \lambda_{\mathsf{comp}_j,0}, \lambda_{\mathsf{comp}_j,1}, \lambda_{\mathsf{comp}_j,2} \in \mathbb{Z}_{2^k}$ (s.t. $\lambda_{\mathsf{comp}_j} = \lambda_{\mathsf{comp}_j,0} \oplus \lambda_{\mathsf{comp}_j,1} \oplus \lambda_{\mathsf{comp}_j,2} \mod 2$) and sets $\mathsf{m}_{\mathsf{comp}_j} = \mathsf{comp}_j \oplus \lambda_{\mathsf{comp}_j} \mod 2$ for $j = 0, \ldots, 2^h - 2$.

4. $\mathcal{F}_{\mathsf{Comp}}$ sets each party's shares (for $j = 0, \ldots, 2^h - 2$) as follows:

$$
\begin{aligned}
[\mathsf{comp}_j]_{2,0} &= (\mathsf{m}_{\mathsf{comp}_j},\ \lambda_{\mathsf{comp}_j,1},\ \lambda_{\mathsf{comp}_j,2}) \\
[\mathsf{comp}_j]_{2,1} &= (\mathsf{m}_{\mathsf{comp}_j},\ \lambda_{\mathsf{comp}_j,2},\ \lambda_{\mathsf{comp}_j,0}) \\
[\mathsf{comp}_j]_{2,2} &= (\mathsf{m}_{\mathsf{comp}_j},\ \lambda_{\mathsf{comp}_j,0},\ \lambda_{\mathsf{comp}_j,1}) \\
[\mathsf{comp}_j]_{2,3} &= (\lambda_{\mathsf{comp}_j,0},\ \lambda_{\mathsf{comp}_j,1},\ \lambda_{\mathsf{comp}_j,2})
\end{aligned}
$$

5. $\mathcal{F}_{\mathsf{Comp}}$ sends $\{[\mathsf{comp}_j]_{2,i}\}_{j=0}^{2^h-2}$ to P_i for $i = 0, \ldots, 3$.

$\mathcal{F}_{\mathsf{PathEval}}$ - (path evaluation)

1. Each party P_i sends the message $(\mathsf{PathEval}, \{[\mathsf{comp}_j]_{2,i}\}_{j=0}^{2^h-2}, \{[\mathsf{leafVal}_{j'}]_{2^k,i}\}_{j'=0}^{2^h-1}, \delta)$ to $\mathcal{F}_{\mathsf{PathEval}}$ for $i = 0, \ldots, 3$.

2. After receiving the messages, $\mathcal{F}_{\mathsf{PathEval}}$ reconstructs $\{\mathsf{comp}_j\}_{j=0}^{2^h-2}$ and $\{\mathsf{leafVal}_{j'}\}_{j'=0}^{2^h-1}$ by using the received messages. Then, $\mathcal{F}_{\mathsf{PathEval}}$ computes $\mathsf{res} = \mathsf{leafVal}_{j'}$ where j' s.t. $\wedge_{\ell=0}^{h-1}(j'|_\ell == \mathsf{comp}_{\delta(j',\ell)}) = 1$.

3. $\mathcal{F}_{\mathsf{PathEval}}$ generates the random values $\lambda_{\mathsf{res}}, \lambda_{\mathsf{res},0}, \lambda_{\mathsf{res},1}, \lambda_{\mathsf{res},2} \in \mathbb{Z}_{2^k}$ (s.t. $\lambda_{\mathsf{res}} = \lambda_{\mathsf{res},0} + \lambda_{\mathsf{res},1} + \lambda_{\mathsf{res},2} \mod 2^k$) and sets $\mathsf{m}_{\mathsf{res}} = \mathsf{res} + \lambda_{\mathsf{res}} \mod 2^k$.

4. $\mathcal{F}_{\mathsf{PathEval}}$ sets each party's shares as follows:

$$[\mathsf{res}]_{2^k,0} = (\mathsf{m}_{\mathsf{res}},\ \lambda_{\mathsf{res},1},\ \lambda_{\mathsf{res},2})$$
$$[\mathsf{res}]_{2^k,1} = (\mathsf{m}_{\mathsf{res}},\ \lambda_{\mathsf{res},2},\ \lambda_{\mathsf{res},0})$$
$$[\mathsf{res}]_{2^k,2} = (\mathsf{m}_{\mathsf{res}},\ \lambda_{\mathsf{res},0},\ \lambda_{\mathsf{res},1})$$
$$[\mathsf{res}]_{2^k,3} = (\lambda_{\mathsf{res},0},\ \lambda_{\mathsf{res},1},\ \lambda_{\mathsf{res},2})$$

5. $\mathcal{F}_{\mathsf{PathEval}}$ sends $[\mathsf{res}]_{2^k,i}$ to P_i for $i = 0, \ldots, 3$.

Theorem 11. *If* $\mathsf{OpenOne}$ *is secure, then the path evaluation protocol* Π_{PathEval} *(Protocol 22) in the* $\mathcal{F}_{\mathsf{TableShuffle}}$*-hybrid model computes* $\mathcal{F}_{\mathsf{PathEval}}$ *securely with computational security and fairness in the presence of one malicious party.*

Proof. We replace calling $\Pi_{\mathsf{TableShuffle}}$ with invoking the ideal functionality $\mathcal{F}_{\mathsf{TableShuffle}}$. We prove the security of $\Pi_{\mathsf{PathEval}}^{\mathcal{F}_{\mathsf{TableShuffle}}}$. $\Pi_{\mathsf{PathEval}}^{\mathcal{F}_{\mathsf{TableShuffle}}}$ consists of invoking the ideal functionality $\mathcal{F}_{\mathsf{TableShuffle}}$, executing $\mathsf{OpenOne}$, and computations without communications. Therefore, the simulator $\mathcal{S}$ can be composed in the case where a party is corrupted. Hence, $\Pi_{\mathsf{PathEval}}^{\mathcal{F}_{\mathsf{TableShuffle}}}$ satisfies Definition 1. Therefore, Theorem 11 holds. □

Theorem 12. *The PDTE protocol* Π_{PDTE} *in the* $(\mathcal{F}_{\mathsf{FSelection}}, \mathcal{F}_{\mathsf{Comp}}, \mathcal{F}_{\mathsf{PathEval}})$*-hybrid model computes* $\mathcal{F}_{\mathsf{PDTE}}$ *securely with computational security and fairness in the presence of one malicious party.*

Proof. We replace calling $\Pi_{\mathsf{FSelection}}$, Π_{Comp}, and Π_{PathEval} with invoking the ideal functionality $\mathcal{F}_{\mathsf{FSelection}}$, $\mathcal{F}_{\mathsf{Comp}}$, and $\mathcal{F}_{\mathsf{PathEval}}$, respectively. We prove the security of $\Pi_{\mathsf{PDTE}}^{\mathcal{F}_{\mathsf{FSelection}},\mathcal{F}_{\mathsf{Comp}},\mathcal{F}_{\mathsf{PathEval}}}$. $\Pi_{\mathsf{PDTE}}^{\mathcal{F}_{\mathsf{FSelection}},\mathcal{F}_{\mathsf{Comp}},\mathcal{F}_{\mathsf{PathEval}}}$ consists of invoking the ideal functionalities $\mathcal{F}_{\mathsf{FSelection}}$, $\mathcal{F}_{\mathsf{Comp}}$, and $\mathcal{F}_{\mathsf{PathEval}}$, and computations without communications. Therefore, the simulator $\mathcal{S}$ can be composed in the case where a party is corrupted. Hence, $\Pi_{\mathsf{PDTE}}^{\mathcal{F}_{\mathsf{FSelection}},\mathcal{F}_{\mathsf{Comp}},\mathcal{F}_{\mathsf{PathEval}}}$ satisfies Definition 1. Therefore, Theorem 12 holds. □

$\mathcal{F}_{\mathsf{PDTE}}$ - (private decision tree evaluation)

1. Each party P_i sends the message $(\mathsf{PDTE}, \{[\mathsf{attr}_{i'}]_{2^k,i}\}_{i'=0}^{m-1}, \mathcal{T} = (h, \delta, \{[\mathsf{idx}_j]_{2^k,i}\}_{j=0}^{2^h-2}, \{[\mathsf{v}_j]_{2^k,i}\}_{j=0}^{2^h-2}, \{[\mathsf{cond}_j]_{2,i}\}_{j=0}^{2^h-2}, \{[\mathsf{leafVal}_{j'}]_{2^k,i}\}_{j'=0}^{2^h-1})$ to $\mathcal{F}_{\mathsf{PDTE}}$ for $i = 0, \ldots, 3$.

2. After receiving the messages, $\mathcal{F}_{\mathsf{PDTE}}$ reconstructs $\{\mathsf{idx}_j\}_{j=0}^{2^h-2}$, $\{\mathsf{attr}_{i'}\}_{i'=0}^{m-1}$, $\{\mathsf{v}_j\}_{j=0}^{2^h-2}$, $\{\mathsf{cond}_j\}_{j=0}^{2^h-2}$ and $\{\mathsf{leafVal}_{j'}\}_{j'=0}^{2^h-1}$ by using the received messages.

3. $\mathcal{F}_{\mathsf{PDTE}}$ computes as follows for $j = 0, \ldots, 2^h - 2$:

   - If $\mathsf{cond}_j = 0$, $\mathcal{F}_{\mathsf{PDTE}}$ computes the comparison result bit $\mathsf{comp}_j = 1$ if $\mathsf{attr}_{\mathsf{idx}} == \mathsf{v}_j$ else $\mathsf{comp}_j = 0$.
   - If $\mathsf{cond}_j = 1$, $\mathcal{F}_{\mathsf{PDTE}}$ computes the comparison result bit $\mathsf{comp}_j = 1$ if $\mathsf{attr}_{\mathsf{idx}} < \mathsf{v}_j$ else $\mathsf{comp}_j = 0$.

4. $\mathcal{F}_{\mathsf{PDTE}}$ computes $\mathsf{res} = \mathsf{leafVal}_{j'}$ where j' s.t. $\wedge_{\ell=0}^{h-1}(j'|_\ell == \mathsf{comp}_{\delta(j',\ell)}) = 1$.

5. $\mathcal{F}_{\mathsf{PDTE}}$ generates the random values $\lambda_{\mathsf{res}}, \lambda_{\mathsf{res},0}, \lambda_{\mathsf{res},1}, \lambda_{\mathsf{res},2} \in \mathbb{Z}_{2^k}$ (s.t. $\lambda_{\mathsf{res}} = \lambda_{\mathsf{res},0} + \lambda_{\mathsf{res},1} + \lambda_{\mathsf{res},2} \mod 2^k$) and sets $\mathsf{m}_{\mathsf{res}} = \mathsf{res} + \lambda_{\mathsf{res}} \mod 2^k$.

6. $\mathcal{F}_{\mathsf{PDTE}}$ sets each party's shares as follows:

$$
\begin{aligned}
[\mathsf{res}]_{2^k,0} &= (\mathsf{m}_{\mathsf{res}},\ \lambda_{\mathsf{res},1},\ \lambda_{\mathsf{res},2}) \\
[\mathsf{res}]_{2^k,1} &= (\mathsf{m}_{\mathsf{res}},\ \lambda_{\mathsf{res},2},\ \lambda_{\mathsf{res},0}) \\
[\mathsf{res}]_{2^k,2} &= (\mathsf{m}_{\mathsf{res}},\ \lambda_{\mathsf{res},0},\ \lambda_{\mathsf{res},1}) \\
[\mathsf{res}]_{2^k,3} &= (\lambda_{\mathsf{res},0},\ \lambda_{\mathsf{res},1},\ \lambda_{\mathsf{res},2})
\end{aligned}
$$

7. $\mathcal{F}_{\mathsf{PDTE}}$ sends $[\mathsf{res}]_{2^k,i}$ to P_i for $i = 0, \ldots, 3$.

From Theorems 1 to 12, if the pseudo-random function F_L and fair 4PC building blocks are secure, our PDTE protocol Π_{PDTE} computes $\mathcal{F}_{\mathsf{PDTE}}$ securely with computational security and fairness in the presence of one malicious party.

4.10 Summary

In this chapter, we proposed the PDTE protocol with constant rounds using (only) the semi-honest secure SS-3PC over the ring for the first time. We also proposed more efficient PDTE protocols with constant rounds by using (only) the semi-honest secure SS-3PC over the field than the naive construction. Furthermore, we proposed the PDTE protocol with constant rounds using (only) the maliciously secure SS-4PC over the ring for the first time. Our schemes provide the PDTE efficiently even where the communication environment has a large latency and limited communication bandwidth. The generalization of the proposed protocol to the N-party protocol and improvement of its security (e.g., information-theoretic security) are open problems to be addressed in the future.

Chapter 5

Secure Iris Authentication via robust SS-MPC[16]

5.1 Introduction

5.1.1 Background

The strongest security notion is GOD, which ensures that all parties including a malicious adversary learn the correct outputs regardless of the attacker's behaviour. There are two main types of GOD: *traditional robustness* [17] and *private robustness* [14, 16]. The former ensures that all parties learn the correct outputs, but there is a possibility that the parties' inputs may be known to a honest party. The latter ensures that all parties learn the correct outputs while keeping information on parties' inputs secret from all parties. Hence, the latter is a stronger security notion than the former. In particular, the existing protocols with private robustness achieve the cheating detection and cheater identification implicitly because the private robustness in these protocols is achieved by detecting the cheating and removing the values sent from the identified cheater without revealing the parties' inputs. For example, Fantastic Four [16] is capable of detecting cheating deterministically, but the identification of the cheater is achieved probabilistically depending on a statistical parameter. FLASH [14] is capable of both detecting cheating and identifying the cheater, deterministically. The performance of the protocols that realize probabilistic cheating detection or probabilistic cheater identification degrades as the statistical parameter is increased. Hence, if strong security is to be achieved, the protocols that realize deterministic cheating detection and deterministic cheater identification are preferable.

The practical advantage of MPC with GOD is that it is secure against a malicious adversary who performs denial of service (DoS) attacks. Real applications or services using MPC schemes that achieve security with abort or fairness are not secure against DoS attacks because it is easy for the adversary to shut them down by sending incorrect values in MPC.

[16]This chapter is based on "Client-aided Robust Bit-composition Protocol with Deterministic Cheater Identification in Standard Model" [21], by the same author, which appeared in the Journal of Information Processing, Volume 29, Copyright(C)2021 Information Processing Society of Japan. The content of this chapter corresponds to references [20, 21] relevant to the requirements for completion.

In comparison, real applications or services using MPC schemes with GOD are robust against such DoS attacks.

In the SS-MPC protocol, each party computes any function represented as a binary, arithmetic or mixed circuit (which is composed of binary and arithmetic circuits) by using shares locally and communicating among the parties. In particular, the secure three-party or four-party protocol [10, 12–14, 16, 17] has gained attention in recent years because it can achieve a high throughput even when it computes a complex function represented as mixed circuits.

An efficient *share conversion* protocol is required when the SS-MPC protocol computes a complex function represented as mixed circuits. Let $\mathbb{Z}_2$ and $\mathbb{Z}_{2^k}$ be residue ring modulo 2 and 2^k, respectively. For example, the addition of the shares on the arithmetic ring $\mathbb{Z}_{2^k}$ is for free. However, the exclusive OR (XOR) operation of the shares of bit on $\mathbb{Z}_{2^k}$ requires communication. In comparison, the XOR operation for the shares on the binary ring $\mathbb{Z}_2$ is for free. However, the addition of shares on $\mathbb{Z}_2$ requires communication. Therefore, converting shares with changing the modulus according to the type of circuits can reduce the communication cost of the entire computation.

The bit-composition protocol is a protocol that converts a k-dimensional vector with shares of $x_j \in \{0,1\} (j = 0, \ldots, k-1)$ on $\mathbb{Z}_2$ (or a binary field $\mathbb{F}_2$) to shares of $x (= \sum_{j=0}^{k-1} 2^j \cdot x_j)$ on $\mathbb{Z}_{2^k}$ (or a finite field $\mathbb{F}_q$ s.t. q is a prime number). In particular, the bit conversion protocol is a protocol that converts shares of $x \in \{0,1\}$ on $\mathbb{Z}_2$ (or $\mathbb{F}_2$) to shares of x on $\mathbb{Z}_{2^k}$ (or $\mathbb{F}_q$). Note that these conversion protocols that do not require changing the modulus are easily achievable. However, it is hard to construct these protocols by changing the modulus. An existing maliciously secure bit-composition protocol achieves security with abort [10, 96], fairness [12, 13], or GOD [14, 16, 17]. Byali et al. proposed a bit conversion protocol with private robustness independent of a statistical parameter in [14]. The bit-composition protocol with private robustness independent of a statistical parameter can be achieved by running the bit conversion protocol with private robustness independent of statistical parameter [14] in parallel k times and using shares locally. However, the bit conversion protocol in [14] uses a commitment scheme that uses a collision-resistant hash function. Therefore, its security is proved only in the random oracle model (ROM). In the ROM, the collision-resistant hash function is replaced by an ideal random function. In comparison, there are no such replacements in the standard model. Therefore, the ROM is a stronger assumption than the standard model.

From a practical viewpoint, an MPC scheme that computes complex functions represented as mixed circuits and achieves private robustness independent of a statistical parameter is preferred. Hence, a bit-composition protocol with private robustness independent of a statistical parameter is required. However, there are no such protocols for the standard model.

5.1.2 Our Approach

In this chapter, we propose a client-aided bit-composition protocol with private robustness independent of a statistical parameter that is provably secure in the standard model. The proposed scheme is based on a maliciously secure four-party computation with one corrup-

Table 5.1: Comparison between existing maliciously secure bit-composition protocols with one corruption and proposed protocol *(Rounds: number of communication rounds, Comm.: (amortized) communication bits per all parties, k: bit length of modulus, N: number of parties, H: number of clients, t: number of malicious corruptions in protocol, t_{p}: number of malicious corruptions in parties, t_{c}: number of malicious corruptions in clients, Std.: standard model, ROM: random oracle model)*

Scheme	Threshold	Property	Deterministic cheating detection	Deterministic cheater identification	Model	Offline phase		Online phase	
						Rounds	Comm.	Rounds	Comm.
ABY3 [10] + [9]	$(t,N)=(1,3)$	abort	(probabilistic)	(not achievable)	Std.	3	$12k\log_2 k+12k$	$1+\log_2 k$	$9k\log_2 k+9k$
BLAZE [13]	$(t,N)=(1,3)$	fairness	(probabilistic)	(not achievable)	Std.	5	$9k^2$	1	$4k^2$
Trident [12]	$(t,N)=(1,4)$	fairness	✓	(not achievable)	Std.	2	$3k^2+k$	1	$3k$
FLASH [14]	$(t,N)=(1,4)$	private robustness	✓	✓	ROM	2	$4k^2$	3	$10k^2$
SWIFT [17]	$(t,N)=(1,4)$	traditional robustness	✓	(not achievable)	Std.	2	$3k^2+4k$	1	$3k^2$
Fantastic Four [16]	$(t,N)=(1,4)$	private robustness	✓	(probabilistic)	Std.	2	$8k^2+8k$	1	$8k$
This Work (Protocol 27)	$(t,N,H)=(1,4,3)$	private robustness	✓	✓	Std.	1	$24k^2+24k$	1	$8k$
This Work (Protocol 28)	$(t_{\mathsf{p}},N,H)=(1,4,1)$	private robustness	✓	✓	Std.	1	$12k^2+12k$	1	$8k$
This Work (Protocol 30)	$t_{\mathsf{p}}(2t_{\mathsf{p}}+1)<N$, $2t_{\mathsf{c}}+1<H$	private robustness	✓	✓	Std.	1	$(t_{\mathsf{c}}+1)(2t_{\mathsf{p}}+1)k^2$ $+(t_{\mathsf{c}}+1)(2t_{\mathsf{p}}+1)k$	1	$N(N-(2t_{\mathsf{p}}+1))(t_{\mathsf{p}}+1)k$

tion [14] with three additional clients who assist the parties in the calculation. Note that these clients provide an auxiliary input only in the preprocessing phase. They do not input any secret values, do not compute any values, and do not learn the output during the actual computation. That is, we propose a maliciously secure seven-party bit-composition protocol with one corruption that achieves private robustness independent of a statistical parameter and is provably secure in the standard model for the first time. Our scheme can improve the efficiency and security of computation for complex functions represented as mixed circuits.

We also propose a secure computation protocol for the Hamming distance by modifying the proposed bit-composition protocol. Our protocol for the Hamming distance can be useful for secure iris authentication applications.

Furthermore, we extend our protocol with a constant number of parties and clients to one with an arbitrary number of parties and clients. In our extended protocol, the parameters (i.e., the number of parties and clients) are flexible so that it fits in many situations.

Table 5.1 shows a comparison of the communication cost between the proposed scheme and the existing maliciously secure bit-composition protocols. This table shows that only our scheme achieves private robustness composed of deterministic cheating detection and cheater identification in the standard model. Furthermore, it shows that the number of communication rounds of our scheme is the lowest compared with other previous schemes.

5.1.3 Related Work

5.1.3.1 Typical Method for Reducing Communication Cost

In MPC protocols, it is important to reduce the communication cost. For example, introducing entities that help with part of the computation [97] is known as a typical methodology for reducing the communication cost. The offline-online paradigm [12–14, 71] is another method. The offline-online paradigm divides the MPC protocol into an offline phase and online phase. In the offline phase, the protocol processes part of the computation that can

be done independently of the parties' inputs. In the online phase, the protocol processes the rest of the computation with the parties' inputs. The offline-online paradigm can reduce the communication cost of the online phase even if it increases the communication cost of the offline phase and the whole computation. The offline-online paradigm is suited to MPC applications that focus on the response time of queries. However, to the best of our knowledge, simultaneously realizing private robustness and the communication cost reduction of the bit-composition protocol by these methodologies has not been proposed.

5.1.3.2 Bit-composition Protocol

There are two main ways to realize the existing bit-composition protocol: using an adder circuit and executing a bit conversion protocol in parallel. The former approach is used in [10]. It can change the communication complexity according to the adder circuit used in the protocol. It can reduce the total communication volume, but it is not a constant-round protocol. Therefore, it is not suited to a network with low latency.

The latter one is used in [12–14,16,96]. It is a constant-round protocol. Hence, it is suited to a network with low latency. In particular, this approach is compatible with the offline-online paradigm. In [12,16,17,96], the shares of the random bit $r_j \in \{0,1\}$ $(j = 0, \ldots, k-1)$ on $\mathbb{Z}_2$ and $\mathbb{Z}_{2^k}$ (or $\mathbb{F}_q$) are prepared in the offline phase. Then, in the online phase, the communication complexity can be reduced to $O(k)$ by using the shares of the random bit generated in the offline phase, even if the bit conversion protocol is executed in parallel. Among the existing works, only the bit conversion protocol in [14] achieves private robustness independent of a statistical parameter (in ROM). Therefore, to the best of our knowledge, there is no bit-composition protocol that simultaneously achieves private robustness independent of a statistical parameter in the standard model and $O(k)$ communication complexity in the online phase.

5.1.3.3 MPC with GOD

Most of the existing protocols have used a broadcast channel or an expensive asymmetric-key primitive [15,98,99]. In recent years, secure four-party protocols achieving GOD without the broadcast channel or the expensive asymmetric-key primitive are proposed [14,16,17]. Byali et al. proposed a secure four-party protocol with GOD (private robustness independent of a statistical parameter) that does not use the broadcast channel or an expensive asymmetric-key primitive in [14]. However, it uses a commitment scheme of which the security is proved in ROM. Therefore, the security of the whole protocol is also proved in ROM. To the best of our knowledge, there has been no proposed general MPC protocol that achieves private robustness independent of a statistical parameter in the standard model without a broadcast channel or an expensive asymmetric-key primitive nor even a specific protocol that achieves it.

5.2 Preliminaries in Chapter 5

5.2.1 2-out-of-4 Replicated Secret Sharing Scheme ((2,4)-RSS)

We use the (2,4)-RSS (also known as mirrored sharing) in [14]. We denote the (2,4)-RSS's shares of x on $\mathbb{Z}_{2^k}$ as $[x]$. Each P_i has share $[x]_i$ as follows, where $\sigma_x, \sigma_x^1, \sigma_x^2 \in \mathbb{Z}_{2^k}$, and $\mu_x, \mu_x^1, \mu_x^2 \in \mathbb{Z}_{2^k}$ such that $\mu_x = x + \sigma_x \bmod 2^k$, $\sigma_x = \sigma_x^1 + \sigma_x^2 \bmod 2^k$, and $\mu_x = \mu_x^1 + \mu_x^2 \bmod 2^k$:

- P_0's share: $[x]_0 = (\sigma_x^1, \mu_x^1, \mu_x^2)$
- P_1's share: $[x]_1 = (\sigma_x^1, \sigma_x^2, \mu_x^1)$
- P_2's share: $[x]_2 = (\sigma_x^2, \mu_x^1, \mu_x^2)$
- P_3's share: $[x]_3 = (\sigma_x^1, \sigma_x^2, \mu_x^2)$

We denote the (2,4)-RSS's shares of x on $\mathbb{Z}_2$ as $[x]^{\mathsf{B}}$. We note that P_0, P_1, P_2 and P_3 correspond to E_1, V_1, E_2 and V_2 in [14], respectively.

5.2.2 $(N - 2t_{\mathsf{p}})$-out-of-N Replicated Secret Sharing Scheme ($(N - 2t_{\mathsf{p}}, N)$-RSS)

Protocol 23 $x \leftarrow \pi_{\mathsf{mboOneParty}}([\![x]\!]^{\mathsf{B}}, \ell)$

Input: $[\![x]\!]^{\mathsf{B}}$ (where $x, x_j \in \mathbb{Z}_2$ and $x = x_0 \oplus \cdots \oplus x_{N-1} \mod 2$)
Output: P_ℓ obtains x.

1: **for** $j = 0, \ldots, \ell - 1, \ell + 2t_{\mathsf{p}} + 1, \ldots, N - 1$ **do in parallel**
2: Each P_i sends x_j as $m_{j,i}$ to P_ℓ for $i = j-(2t_{\mathsf{p}}+1), \ldots, j-t_{\mathsf{p}}$. // 1 round and $t_{\mathsf{p}} + 1$ bits
3: Each P_i sends the hashed value of x_j, $h_{j,i} = \mathcal{H}(x_j) = \mathcal{H}(m_{j,i})$ for $i = j - t_{\mathsf{p}}, \ldots, j$. // 1 round & $t_{\mathsf{p}}|h_{i,j}|$ bits
4: P_ℓ computes the hashed value of $m_{j,i}$, $h_{j,i}$ for $i = j - (2t_{\mathsf{p}} + 1), \ldots, j - t_{\mathsf{p}}$. Then, P_ℓ chooses $m_j \in \{m_{j,i}\}_{i=j-(2t_{\mathsf{p}}+1)}^{j-t_{\mathsf{p}}}$ as the correct value x_j if $t_{\mathsf{p}} + 1$ or more of the hashed values in $\{h_{j,i}\}_{i=0}^{N-1}$ match the hashed value of m_j, $\mathcal{H}(m_j)$.
5: **end for**
6: P_ℓ computes $x = x_0 \oplus \cdots \oplus x_{N-1} \mod 2$ by using $[\![x]\!]_\ell^{\mathsf{B}}$ and the chosen values as correct values in the previous steps.

Protocol 24 $x \leftarrow \pi_{\mathsf{mbo}}([\![x]\!]^{\mathsf{B}})$

Input: $[\![x]\!]^{\mathsf{B}}$ (where $x, x_i \in \mathbb{Z}_2$ and $x = x_0 \oplus \cdots \oplus x_{N-1} \mod 2$)
Output: All parties obtain x.

1: **for** $i = 0, \ldots, N - 1$ **do in parallel**

2: All parties run $\pi_{\mathsf{mboOneParty}}([\![x]\!]^{\mathsf{B}}, i)$ and P_i gets x. // 1 round & $N(N-(2t_{\mathsf{p}}+1))(t_{\mathsf{p}}+1)$ bits

3: **end for**

Let N and t_{p} be the number of parties and the corruptions in the parties. We use the $(N-2t_{\mathsf{p}}, N)$-RSS where $t_{\mathsf{p}}(2t_{\mathsf{p}}+1) < N$ (i.e., at most $O(\sqrt{N})$ corruptions similar to the secure N-party computation protocol proposed by Byali et al. [98] or Chandran et al. [100]). We denote the $(N-2t_{\mathsf{p}}, N)$-RSS's shares of x on $\mathbb{Z}_{2^k}$ as $[\![x]\!]$. Each P_i has share $[\![x]\!]_i = (x_i, \ldots, x_{i+2t_{\mathsf{p}}})$ where $x = x_0 + \cdots + x_{N-1} \mod 2^k$. That is, each P_i has $(2t_{\mathsf{p}}+1)$ of these x_j $(j = 0, \ldots, N-1)$, ranging from x_i to $x_{i+2t_{\mathsf{p}}}$ as shares. In other words, each x_j $(j = 0, \ldots, N-1)$ is possessed by $(2t_{\mathsf{p}}+1)$ parties. Therefore, if $t_{\mathsf{p}}(2t_{\mathsf{p}}+1) < N$ holds, then the secret value cannot be reconstructed from shares of $(N-2t_{\mathsf{p}}, N)$-RSS even if t_{p} corruptions occur. We also denote the $(N-2t_{\mathsf{p}}, N)$-RSS's shares of x on $\mathbb{Z}_2$ as $[\![x]\!]^{\mathsf{B}}$ as in Protocols 23 (multiparty binary shares opening (mbo) protocol for one party, $\pi_{\mathsf{mboOneParty}}$) and 24 (multiparty binary shares opening protocol, π_{mbo}).

Each party P_i (for $i = 0, \ldots, N-1$) performs the share addition, $[\![x+y]\!] = [\![x]\!] + [\![y]\!]$ by setting $[\![x+y]\!]_i = (x_i + y_i \mod 2^k, \ldots, x_{i+2t_{\mathsf{p}}} + y_{i+2t_{\mathsf{p}}} \mod 2^k)$. If parties perform the scalar addition, $[\![c+x]\!] = c + [\![x]\!]$, where $x, x_i \in \mathbb{Z}_{2^k}$, $x = \sum_{i=0}^{N-1} x_i \mod 2^k$, and $c \in \mathbb{Z}_{2^k}$ is public, the parties who obtain x_0 replace x_0 (in their shares) with $x'_0 = x_0 + c \mod 2^k$. Each party performs the scalar multiplication, $[\![c \cdot x]\!] = c \cdot [\![x]\!]$, where $c \in \mathbb{Z}_{2^k}$, by setting $[\![c \cdot x]\!]_i = (c \cdot x_i \mod 2^k, \ldots, c \cdot x_{i+2t_{\mathsf{p}}} \mod 2^k)$. Therefore, the linearity of shares of $(N-2t_{\mathsf{p}}, N)$-RSS on $\mathbb{Z}_{2^k}$ holds, i.e., $[\![c_0 \cdot x + c_1 \cdot y]\!] = c_0 \cdot [\![x]\!] + c_1 \cdot [\![y]\!]$, where $c_0, c_1 \in \mathbb{Z}_{2^k}$. The linearity of shares of $(N-2t_{\mathsf{p}}, N)$-RSS on $\mathbb{Z}_2$ holds in the same way as $(N-2t_{\mathsf{p}}, N)$-RSS on $\mathbb{Z}_{2^k}$.

We explain how to reconstruct the secret value from shares of $(N-2t_{\mathsf{p}}, N)$-RSS. We describe the opening protocol of $(N-2t_{\mathsf{p}}, N)$-RSS on $\mathbb{Z}_2$, Protocol 24, because we use it only on $\mathbb{Z}_2$ (not $\mathbb{Z}_{2^k}$) in our protocol. Protocol 24 is the opening protocol on $\mathbb{Z}_2$ for all parties. It is realized by running the opening protocol on $\mathbb{Z}_2$ for one party (Protocol 23) in parallel. Protocols 23 and 24 achieve private robustness independent of a statistical parameter because of the majority voting in Line 4 of Protocol 23[17].

Protocol 23 requires 1 round and $(N-(2t_{\mathsf{p}}+1))(t_{\mathsf{p}}+1+t_{\mathsf{p}}|h_{i,j}|)$ bits as communication bits. Note that we can ignore the communication cost of Step 3 of Protocol 23 when running multiple instances. Hence, Protocol 23 requires 1 round and $(N-(2t_{\mathsf{p}}+1))(t_{\mathsf{p}}+1)$ bits as amortized communication bits. Therefore, Protocol 24 requires 1 round and $N(N-(2t_{\mathsf{p}}+1))(t_{\mathsf{p}}+1)$ bits as amortized communication bits.

Protocol 25 $[\![x]\!] \leftarrow \pi_{\mathsf{share}}(x,\ P_\ell,\ \mathsf{vid}, F)$

Input: Input value $x \in \mathbb{Z}_2$, input dealer P_ℓ, unique identifier vid, pseudo-random function $F : \{0,1\}^\kappa \times \{0,1\}^\kappa \to \mathbb{Z}_2$

Output: $[\![x]\!]$

[17] At Line 4 in Protocol 23, the security is not compromised even if P_i $(i = j - t_{\mathsf{p}}, \ldots, j)$ sends the raw value, i.e., $m_{j,i}$. The reason why P_i $(i = j - t_{\mathsf{p}}, \ldots, j)$ sends the hashed value, $\mathcal{H}(m_{j,i})$, is just to reduce the cost of communication volume.

1: **for** $i = 0, \ldots, N-1$ **do in parallel**
2: Each party P_i computes $r_j = F(\mathsf{seed}_j,\ \mathsf{vid})$ where $r_j \in \mathbb{Z}_2$ for $j = i, \ldots, i + 2t_{\mathsf{p}}$.
3: Each party P_i sets $[\![r]\!]^{\mathsf{B}}_i = (r_j, \ldots, r_{j+2t_{\mathsf{p}}})$ where $r = r_0 \oplus \cdots \oplus r_{N-1} \mod 2$ and $j = i, \ldots, i + 2t_{\mathsf{p}}$.
4: **end for**
5: An input dealer P_ℓ generates random values $x_j \in \mathbb{Z}_2$ for $j = 1, \ldots, N-1$.
6: P_ℓ sets $x_0 = x \oplus x_1 \oplus \cdots \oplus x_{N-1} \mod 2$.
7: **for** $i = 0, \ldots, N-1$ **do in parallel**
8: P_ℓ sends $[\![x]\!]^{\mathsf{B}}_i = (x_i, \ldots, x_{i+2t_{\mathsf{p}}})$ to P_i `// 1 round & (N − 1)(2t_p + 1) bits`
9: **end for**
10: $[\![x \oplus r]\!]^{\mathsf{B}} = [\![x]\!]^{\mathsf{B}} \oplus [\![r]\!]^{\mathsf{B}}$
11: All parties run $\pi_{\mathsf{mbo}}([\![x \oplus r]\!]^{\mathsf{B}})$ and gets $x \oplus r$. `// 1 round & N(N − (2t_p + 1))(t_p + 1) bits`
12: All parties exchange their $(x \oplus r)$ and choose the correct $(x \oplus r)$ by the majority voting. If it is not possible to determine the correct value by majority voting, then P_ℓ and its initial input values are removed.
13: $[\![x]\!]^{\mathsf{B}} = (x \oplus r) \oplus [\![r]\!]^{\mathsf{B}}$

Note that we do not use the sharing protocol of $(N - 2t_{\mathsf{p}}, N)$-RSS in our protocol. However, we describe it over $\mathbb{Z}_2$ in Protocol 25 for completeness.

We assume that each P_i $(i = 0, \ldots, N-1)$ obtains $(\mathsf{seed}_i, \ldots, \mathsf{seed}_{i+2t_{\mathsf{p}}})$ where $\mathsf{seed}_i \in \{0,1\}^\kappa$. We also assume that $(\mathsf{seed}_i, \ldots, \mathsf{seed}_{i+2t_{\mathsf{p}}})$ are given to each P_i only once by a trusted third party or MPC-based random value generation during the initialization process.

From Lines 1 to 4 in Protocol 25, each party computes the shares of random bit r, $[\![r]\!]^{\mathsf{B}}$ to mask the input value. Then, the input dealer P_ℓ computes the shares of input value $[\![x]\!]^{\mathsf{B}}$ and sends $[\![x]\!]^{\mathsf{B}}_i$ to P_i from Lines 5 to 9. After that, all parties compute $[\![x \oplus r]\!]^{\mathsf{B}}$ at Line 10 and get $(x \oplus r)$ by π_{mbo} at Line 11. At Line 12, all parties exchange $(x \oplus r)$ and choose the correct $(x \oplus r)$ by majority voting. However, if P_ℓ is the corrupted party, P_ℓ can cause so much corruptions that $t_{\mathsf{p}}(2t_{\mathsf{p}} + 1) < N$ does not hold, indirectly, by sending the incorrect different values to each party in the previous Lines. Hence, if the majority voting does not work well, the parties except P_ℓ identify P_ℓ as the corrupted party and remove it and its initial inputs. Finally, all parties compute $[\![x]\!]^{\mathsf{B}} = (x \oplus r) \oplus [\![r]\!]^{\mathsf{B}}$ at Line 13 and get the shares of the input value $[\![x]\!]^{\mathsf{B}}$.

5.2.3 Secure Four-party Computation with One Corruption

We use the same addition of shares as [14]. We denote the addition of shares as $[x] + [y]$. We also use the same scalar addition and scalar multiplication of shares as [14]. We denote the scalar addition and scalar multiplication of shares as $c + [x]$ and $c \cdot [x]$, where $c \in \mathbb{Z}_{2^k}$. We denote the operations on $\mathbb{Z}_2$ in the same way as on $\mathbb{Z}_{2^k}$. Note that we use the same notation for operations of scalars and those of the shares to keep the description simple.

We also use the same four-party binary shares opening (bo) protocol (i.e., output computation protocol) as [14]. The bo protocol of [14] (not the bit conversion protocol of [14]) achieves private robustness independent of a statistical parameter in the standard model

because it uses the collision-resistant hash function only for reducing the communication cost but does not use the commitment scheme based on the collision-resistant hash function. We denote the bo protocol on $\mathbb{Z}_2$ as π_{bo}. π_{bo} takes a share $[x]^{\mathsf{B}}$ as input and outputs $x \in \mathbb{Z}_2$. We denote $x \leftarrow \pi_{\mathsf{bo}}([x]^{\mathsf{B}})$ when it is called. Protocol π_{bo} needs one round and 8 bits as the (amortized) communication cost, respectively.

5.2.4 Definition of Security

Let $\mathsf{view}_i^{\pi}(\vec{x})$ be the view of P_i while running protocol π on inputs $\vec{x}$. It consists of its inputs $\vec{x}$, all the messages received by P_i, and an internal random coin. Let $\mathsf{output}^{\pi}(\vec{x})$ be the outputs of all parties while running protocol π on inputs $\vec{x}$.

Definition 2 (Perfect Security in the Presence of One Malicious Corrupted Party)**.** *Let $f : (\{0,\ 1\}^*)^7 \to (\{0,\ 1\}^*)^7$ be a deterministic 7-ary functionality and let π be a protocol. We say that π computes f with perfect security in the presence of one malicious corrupted party for f if there exists probabilistic polynomial-time algorithm $\mathcal{S}$ such that for every corrupted party $i \in \{0, 1, 2, 3, 4, 5, 6\}$, and every $\vec{x} \in (\{0,1\}^*)^7$ where $|x_0| = |x_1| = |x_2| = |x_3| = |x_4| = |x_5| = |x_6|$:*

$$\{(\mathcal{S}(x_i,\ f_i(\vec{x})),\ f(\vec{x}))\} \equiv \{(\mathsf{view}_i^{\pi}(\vec{x}), \mathsf{output}^{\pi}(\vec{x}))\} \tag{5.1}$$

We note that Definition 2 is modified from the security definition of [8] to adapt the proposed seven-party (i.e., four parties P_0, P_1, P_2 and P_3 and three helper $H_0(= P_4)$, $H_1(= P_5)$ and $H_2(= P_6)$) bit-composition protocol with private robustness independent of a statistical parameter. If the above equation (2) holds with computational indistinguishability, then we say that π computes f with computational security in the presence of one malicious corrupted party.

We use the hybrid model [92] to prove the security of the proposed scheme. Let g be the subfunctionality. We say that protocol π is secure in the g-hybrid model. We denote protocol π secure in the g-hybrid model as π^g.

5.3 Proposal

We propose the new client-aided bit composition protocol with private robustness independent of statistical parameter in §5.3.1. The bit-composition protocol is useful as a subprotocol, but not a useful application. Hence, by modifying our bit-composition protocol, we also propose a new client-aided secure Hamming distance calculation protocol with private robustness independent of statistical parameter as a useful application in §5.3.2. It is useful for the secure iris authentication as a biometric authentication service.

For more details, in §5.3.1, we propose a three bit-composition protocols. In §5.3.1.1, we propose a four-party and three-clients construction with one malicious corruption. We prove that it is secure in §5.3.1.2. We also propose a four-party and one-client construction with one malicious corruption by modifying the four-party and three-clients construction to reduce the number of clients (in §5.3.1.3). In order to allow flexibility in the number of parties and clients, we propose a multi-party and multi-client construction with malicious corrupted parties and clients (in §5.3.1.4).

In §5.3.2, we propose the secure Hamming distance calculation protocol (in §5.3.2.1). We explain how to use our protocol in the secure iris authentication in §5.3.2.2. Then, we modify and extend our secure Hamming distance calculation protocol to reduce the number of clients and allow flexibility in the number of parties and clients in §5.3.2.3.

5.3.1 Client-aided Bit-composition Protocol with Private Robustness Independent of Statistical Parameter

Protocol 26 $\{[r_j]^{\mathsf{B}},\ [r_j]\}_{j=0}^{n-1} \leftarrow \pi_{\mathsf{rRndGen}}(F, F', \mathcal{H}, \mathsf{seed}, \{\mathsf{vid}_j, \mathsf{vid}_j^{(\sigma)}, \mathsf{vid}_j^{(\sigma_1)}, \mathsf{vid}_j^{(\mu_1)}\}_{j=0}^{n-1})$

Input: pseudo-random function $F : \{0,1\}^\kappa \times \{0,1\}^\kappa \to \mathbb{Z}_2$ and $F' : \{0,1\}^\kappa \times \{0,1\}^\kappa \to \mathbb{Z}_{2^k}$, collision-resistant hash function $\mathcal{H}$, shared seed by clients seed, unique identifiers $\{\mathsf{vid}_j, \mathsf{vid}_j^{(\sigma)}, \mathsf{vid}_j^{(\sigma_1)}, \mathsf{vid}_j^{(\mu_1)}\}_{j=0}^{n-1}$ *where* $\mathsf{seed}, \mathsf{vid}_j, \mathsf{vid}_j^{(\sigma)}, \mathsf{vid}_j^{(\sigma_1)}, \mathsf{vid}_j^{(\mu_1)} \in \{0,1\}^\kappa$ (n is a positive integer)

Output: $\{[r_j]^{\mathsf{B}},\ [r_j]\}_{j=0}^{n-1}$ *s.t.* $r_j \in \mathbb{Z}_2$

1: **for** $j = 0, \ldots, n-1$ **do**
2: Each $H_{i'}$ $(i' = 0, 1, 2)$ computes $r_j = F(\mathsf{seed},\ \mathsf{vid}_j)$.
3: Each $H_{i'}$ computes $\sigma_{r_j} = F(\mathsf{seed},\ \mathsf{vid}_j^{(\sigma)})$, $\sigma^1_{r_j} = F(\mathsf{seed}, \mathsf{vid}_j^{(\sigma_1)})$, $\sigma^2_{r_j} = \sigma_{r_j} \oplus \sigma^1_{r_j}$ mod 2.
4: Each $H_{i'}$ computes $\mu_{r_j} = r_j \oplus \sigma_{r_j} \bmod 2$, $\mu^1_{r_j} = F(\mathsf{seed}, \mathsf{vid}_j^{(\mu_1)})$, $\mu^2_{r_j} = \mu_{r_j} \oplus \mu^1_{r_j}$ mod 2.
5: Each $H_{i'}$ computes $\sigma'_{r_j} = F'(\mathsf{seed},\ \mathsf{vid}_j^{(\sigma)})$, $\sigma'^1_{r_j} = F'(\mathsf{seed}, \mathsf{vid}_j^{(\sigma_1)})$, $\sigma'^2_{r_j} = \sigma'_{r_j} - \sigma'^1_{r_j}$ mod 2^k.
6: Each $H_{i'}$ computes $\mu'_{r_j} = r_j + \sigma'_{r_j} \bmod 2^k$, $\mu'^1_{r_j} = F'(\mathsf{seed}, \mathsf{vid}_j^{(\mu_1)})$, $\mu'^2_{r_j} = \mu'_{r_j} - \mu'^1_{r_j}$ mod 2^k.
7: Each $H_{i'}$ sets $[r_j]^{\mathsf{B}}_0 = (\sigma^1_{r_j},\ \mu^1_{r_j},\ \mu^2_{r_j})$, $[r_j]^{\mathsf{B}}_1 = (\sigma^1_{r_j},\ \sigma^2_{r_j},\ \mu^1_{r_j})$, $[r_j]^{\mathsf{B}}_2 = (\sigma^2_{r_j},\ \mu^1_{r_j},\ \mu^1_{r_j})$, $[r_j]^{\mathsf{B}}_3 = (\sigma^1_{r_j},\ \sigma^2_{r_j},\ \mu^2_{r_j})$.
8: Each $H_{i'}$ sets $[r_j]_0 = (\sigma'^1_{r_j},\ \mu'^1_{r_j},\ \mu'^2_{r_j})$, $[r_j]_1 = (\sigma'^1_{r_j},\ \sigma'^2_{r_j},\ \mu'^1_{r_j})$, $[r_j]_2 = (\sigma'^2_{r_j},\ \mu'^1_{r_j},\ \mu'^1_{r_j})$, $[r_j]_3 = (\sigma'^1_{r_j},\ \sigma'^2_{r_j},\ \mu'^2_{r_j})$.
9: **end for**
10: H_0 and H_1 send the set of shares $\{[r_j]^{\mathsf{B}}_i,\ [r_j]_i\}_{j=0}^{n-1}$ to each P_i $(i = 0, 1, 2, 3)$. H_2 computes the hashed value of $\{[r_j]^{\mathsf{B}}_i,\ [r_j]_i\}_{j=0}^{n-1}$, $\mathcal{H}(R_{2,i})$, and send it to each P_i. `// 1 round & 24nk + 24n bits`
11: Let each $R_{0,i}$ and $R_{1,i}$ be the set of shares $\{[r_j]^{\mathsf{B}}_i,\ [r_j]_i\}_{j=0}^{n-1}$ that is sent to P_i from H_0 and H_1, respectively. Each P_i computes the hashed value of $R_{0,i}$ and $R_{1,i}$, $\mathcal{H}(R_{0,i})$ and $\mathcal{H}(R_{1,i})$. If $\mathcal{H}(R_{0,i}) = \mathcal{H}(R_{1,i})$ or $\mathcal{H}(R_{0,i}) = \mathcal{H}(R_{2,i})$, each P_i outputs $R_{0,i}$. If $\mathcal{H}(R_{1,i}) = \mathcal{H}(R_{2,i})$, each P_i outputs $R_{1,i}$.

Protocol 27 $[x] \leftarrow \pi^{\mathcal{F}_{\mathsf{rRndGen}}}_{\mathsf{rBitComp}}(\{[x_j]^{\mathsf{B}}\}_{j=0}^{k-1})$

Input: $\{[x_j]^{\mathsf{B}}\}_{j=0}^{k-1}$ (where $x_j \in \mathbb{Z}_2$)

FUNCTIONALITY 1 ($\mathcal{F}_{\mathsf{rRndGen}}$ - generating two types of random shares).

1. $\mathcal{F}_{\mathsf{rRndGen}}$ receives message $(\mathsf{gen},\ n)$ from clients H_0, H_1 and H_2.
2. $\mathcal{F}_{\mathsf{rRndGen}}$ generates $\{[r_j]^{\mathsf{B}},\ [r_j]\}_{j=0}^{n-1}$ randomly where $r_j \in \{0,1\}$.
3. $\mathcal{F}_{\mathsf{rRndGen}}$ sends $\{[r_j]^{\mathsf{B}},\ [r_j]\}_{j=0}^{n-1}$ to parties P_0, P_1, P_2, and P_3.

FUNCTIONALITY 2 ($\mathcal{F}_{\mathsf{rBitComp}}$ - converting binary share vector on $\mathbb{Z}_2$ to a share on $\mathbb{Z}_{2^k}$).

1. $\mathcal{F}_{\mathsf{rBitComp}}$ receives shares $\{[x_j]^{\mathsf{B}}\}_{j=0}^{k-1}$ from parties P_0, P_1, P_2 and P_3.
2. $\mathcal{F}_{\mathsf{rBitComp}}$ reconstructs x_j and computes $x = \sum_{j=0}^{k-1} 2^j \cdot x_j \bmod 2^k$. Then, $\mathcal{F}_{\mathsf{rBitComp}}$ computes $[x]$ and sends it to parties P_0, P_1, P_2, and P_3.

Output: $[x]$ *s.t.* $x = \sum_{j=0}^{k-1} 2^j \cdot x_j \mod 2^k$

1: (Offline phase)
2: $P_0,\ P_1,\ P_2,\ P_3,\ H_0,\ H_1$ and H_2 invoke $\mathcal{F}_{\mathsf{rRndGen}}$ where $n = k$, then get $\{[r_j]^{\mathsf{B}},\ [r_j]\}_{j=0}^{k-1}$. // π_{rRndGen} requires 1 round & $24k^2 + 24k$ bits
3: (Online phase)
4: Each P_i $(i = 0, 1, 2, 3)$ computes $[c_j]_i^{\mathsf{B}} = [x_j \oplus r_j]_i^{\mathsf{B}} = [x_j]_i^{\mathsf{B}} \oplus [r_j]_i^{\mathsf{B}}$ for $j = 0, \ldots, k-1$.
5: Each P_i gets the value $c_j \leftarrow \pi_{\mathsf{bo}}([c_j]^{\mathsf{B}})$ for $j = 0, \ldots, k-1$ in parallel. // 1 round & $8k$ bits
6: $[x] = \sum_{j=0}^{k-1} 2^j \cdot (c_j + [r_j] - 2 \cdot c_j \cdot [r_j]) \bmod 2^k$.

5.3.1.1 Four-party and Three-clients Construction with One Malicious Corruption

Our scheme is divided into three steps. In the first step (at Lines 1 and 2 in Protocol 27), three clients generate the shares of random bits $r_j \in \mathbb{Z}_2$ $(j = 0, \ldots, k-1)$ on $\mathbb{Z}_2$ and $\mathbb{Z}_{2^k}$, i.e., $\{[r_j]^{\mathsf{B}},\ [r_j]\}_{j=0}^{k-1}$ by running ideal functionality $\mathcal{F}_{\mathsf{rRndGen}}$ (where $n = k$). In the actual protocol, ideal functionality $\mathcal{F}_{\mathsf{rRndGen}}$ is replaced with protocol π_{rRndGen} in Protocol 26. In π_{rRndGen}, three clients generate random values by using the same seed, the unique identifiers $\{\mathsf{vid}_j, \mathsf{vid}_j^{(\sigma)}, \mathsf{vid}_j^{(\sigma_1)}, \mathsf{vid}_j^{(\mu_1)}\}_{j=0}^{n-1}$, and pseudo-random functions F and F' (from Line 1 to 6 in Protocol 26). Then, they set the random values as shares $\{[r_j]^{\mathsf{B}},\ [r_j]\}_{j=0}^{k-1}$ (from Line 7 to 8 in Protocol 26) and send $\{[r_j]_i^{\mathsf{B}},\ [r_j]_i\}_{j=0}^{k-1}$ or the hashed value of them to P_i $(i = 0, 1, 2, 3)$ (at Line 10 in Protocol 26)[18]. Each P_i selects the set of shares that matches two or more matching sets of shares sent from H_0, H_1, and H_2 as correct outputs (at Line 11 in Protocol

[18] At Line 10 in Protocol 26, the security is not compromised even if H_2 sends the raw value, i.e., R_2. The reason why H_2 sends the hashed value, $\mathcal{H}(R_2)$, is just to reduce the cost of communication volume.

26). Each P_i can always get correct random shares $\{[r_j]_i^{\mathsf{B}},\ [r_j]_i\}_{j=0}^{k-1}$ because there can be at most one corrupted client.

In the second step (at Lines 4 and 5 in Protocol 27), each P_i $(i = 0,\ 1,\ 2,\ 3)$ computes $[c_j]_i^{\mathsf{B}} = [x_j \oplus r_j]_i^{\mathsf{B}} = [x_j]_i^{\mathsf{B}} \oplus [r_j]_i^{\mathsf{B}}$ for $j = 0, \ldots, k-1$ and gets the masked values c_j by running the four-party binary shares opening protocol π_{bo}, which is proposed in [14].

In the third step (at Line 6 in Protocol 27), the parties remove mask r_j from c_j by computing $[x_j] = [c_j \oplus r_j] = (c_j - [r_j])^2 = c_j + [r_j] - 2 \cdot c_j \cdot [r_j] \bmod 2^k$. We note that $(c_j)^2 = c_j$ and $(r_j)^2 = r_j$ where $c_j,\ r_j \in \{0, 1\}$. Then, the parties output $[x] = \sum_{j=0}^{k-1} 2^j \cdot (c_j + [r_j] - 2 \cdot c_j \cdot [r_j])$.

5.3.1.2 Security Proof Sketch of Protocol 27

The bit-composition protocol $\pi_{\mathsf{rBitComp}}^{\mathcal{F}_{\mathsf{rRndGen}}}$ in Protocol 27 computes $\mathcal{F}_{\mathsf{rBitComp}}$ with computational security in the presence of one malicious corrupted party because $\pi_{\mathsf{rBitComp}}^{\mathcal{F}_{\mathsf{rRndGen}}}$ consists of $\mathcal{F}_{\mathsf{rRndGen}}$, π_{bo} and local computations. The security of π_{bo} with private robustness independent of a statistical parameter in the standard model is proved in [14] because it uses majority voting and does not use a commitment protocol. Therefore, if we prove that π_{rRndGen} computes $\mathcal{F}_{\mathsf{rRndGen}}$ with computational security in the presence of one malicious corrupted party, we can also prove the security of $\pi_{\mathsf{rBitComp}}^{\mathcal{F}_{\mathsf{rRndGen}}}$ with private robustness independent of a statistical parameter in the standard model.

We prove that protocol π_{rRndGen} in Protocol 26 computes $\mathcal{F}_{\mathsf{rRndGen}}$ with computational security in the presence of one malicious corrupted party. We assume that at most one of $H_{i'}$ $(i^{'} = 0, 1, 2)$ is corrupted by a malicious adversary. In this case, $H_{i'}$ does not learn any information about secret inputs. Therefore, privacy is achieved. The correctness is also achieved with private robustness independent of a statistical parameter by majority voting (i.e., a selection of two or more matching the set of shares which is received) because there is at most one corrupted party. More specifically, either $R_{0,i}$ (that is sent to P_i from H_0) or $R_{1,i}$ (that is sent to P_i from H_1) is always the correct value because there is at most one corrupted client of $H_{i'}$ $(i^{'} = 0, 1, 2)$. Then, at Line 11 in Protocol 26, each P_i can choose either $R_{0,i}$ or $R_{1,i}$ as the correct value by knowing $\mathcal{H}(R_{2,i})$ (that is sent to P_i from H_2) and comparing $\mathcal{H}(R_{0,i})$, $\mathcal{H}(R_{1,i})$ and $\mathcal{H}(R_{2,i})$. Hence, the simulator $\mathcal{S}$ (i.e., the polynomial-time algorithm $\mathcal{S}$ in 2) can be composed.

In another case, we assume that at most one of $P_i (i = 0, 1, 2, 3)$ is corrupted by a malicious adversary. $\mathcal{S}$ can generate random bits $r'_j \in \mathbb{Z}_2$ $(j = 0, \ldots, k-1)$. If the outputs of F and F' are indistinguishable from the random values with computational security, $\mathcal{S}$ can also generate random values $\sigma_{r'_j}, \sigma^1_{r'_j}, \mu^1_{r'_j} \in \mathbb{Z}_2$ and $\sigma'_{r'_j}, \sigma'^1_{r'_j}, \mu'^1_{r'_j} \in \mathbb{Z}_{2^k}$, then set $\sigma^2_{r'_j} = \sigma_{r'_j} \oplus \sigma^1_{r'_j} \bmod 2$, $\mu_{r'_j} = r'_j \oplus \sigma'_{r'_j} \bmod 2$, $\mu^2_{r'_j} = \mu_{r'_j} \oplus \mu^2_{r'_j} \bmod 2$, $\sigma'^2_{r'_j} = \sigma'_{r'_j} - \sigma'^1_{r'_j} \bmod 2^k$, $\mu'_{r'_j} = r'_j + \sigma'_{r'_j} \bmod 2^k$, and $\mu^2_{r'_j} = \mu_{r'_j} - \mu^2_{r'_j} \bmod 2^k$. Then, $\mathcal{S}$ can set $\{[r'_j]^{\mathsf{B}}, [r'_j]\}_{j=0}^{k-1}$ by using these random values in the same way as at Lines 7 and 8 in Protocol 26. Therefore, privacy is achieved. Correctness with private robustness independent of a statistical parameter in the standard model is also achieved because each $P_i (i = 0, 1, 2, 3)$ sends no messages and cannot cheat. Hence, $\mathcal{S}$ can be composed.

To prove that π_{rRndGen} is secure with private robustness independent of a statistical

parameter in the standard model, we prove that the corrupted party or client cannot break private robustness independent of a statistical parameter if $\mathcal{H}$ has the second preimage resistance. In π_{rRndGen}, only the corrupted client can break private robustness because the parties send no messages in π_{rRndGen}. Let the corrupted client be H_{c} ($\mathsf{c} = 0, 1, 2$). If H_{c} would like to cheat and break private robustness, he/she needs to find $R'_{\mathsf{c},i}$ such that $R_{\mathsf{c}-1,i} \neq R'_{\mathsf{c},i}$, $R_{\mathsf{c}+1,i} \neq R'_{\mathsf{c},i}$, $\mathcal{H}(R_{\mathsf{c}-1,i}) = \mathcal{H}(R'_{\mathsf{c},i})$, and $\mathcal{H}(R_{\mathsf{c}+1,i}) = \mathcal{H}(R'_{\mathsf{c},i})$ because breaking private robustness requires that the majority voting does not work. However, if H_{c} (knowing $R_{\mathsf{c}-1,i}$ and $R_{\mathsf{c}+1,i}$) finds such $R'_{\mathsf{c},i}$, H_{c} breaks the second preimage resistance of $\mathcal{H}$. Hence, π_{rRndGen} is secure with private robustness independent of a statistical parameter in the standard model if $\mathcal{H}$ has the second preimage resistance. We note that we can prove that π_{bo} is secure with the private robustness independent of a statistical parameter in the standard model if $\mathcal{H}$ has the second preimage resistance similarly. On the other hand, we emphasize that FLASH [14] needs the collision-resistant hash function that has not only the second preimage resistance but also the preimage resistance and collision resistance because the collision-resistant hash function in FLASH [14] is used as the commitment scheme that achieves the hiding and binding properties. To use a hash function as a commitment scheme, the security of FLASH needs to be proven in the ROM.

Therefore, π_{rRndGen} satisfies Definition 2 and computes $\mathcal{F}_{\mathsf{rRndGen}}$ with computational security in the presence of one malicious corrupted party. We can prove the whole security of $\pi_{\mathsf{rBitComp}}^{\mathcal{F}_{\mathsf{rRndGen}}}$ with private robustness independent of a statistical parameter in the standard model.

5.3.1.3 Four-party and One-client Construction with One Malicious Corrupted Party

Protocol 28 $[x] \leftarrow \pi_{\mathsf{rBitComp5}}(\{[x_j]^{\mathsf{B}}\}_{j=0}^{k-1})$

Input: $\{[x_j]^{\mathsf{B}}\}_{j=0}^{k-1}$ (where $x_j \in \mathbb{Z}_2$)

Output: $[x]$ *s.t.* $x = \sum_{j=0}^{k-1} 2^j \cdot x_j \mod 2^k$

1: (Offline phase)
2: H_0 generates $\{[r_j]^{\mathsf{B}}, [r_j]\}_{j=0}^{k-1}$ randomly and distributes it to the parties. // 1 round & $12k^2 + 12k$ bits
3: (Online phase)
4: Each P_i ($i = 0, 1, 2, 3$) computes $[c_j]_i^{\mathsf{B}} = [x_j \oplus r_j]_i^{\mathsf{B}} = [x_j]_i^{\mathsf{B}} \oplus [r_j]_i^{\mathsf{B}}$ for $j = 0, \ldots, k-1$.
5: Each P_i gets the value $c_j \leftarrow \pi_{\mathsf{bo}}([c_j]^{\mathsf{B}})$ for $j = 0, \ldots, k-1$ in parallel. // 1 round & $8k$ bits
6: $[x] = \sum_{j=0}^{k-1} 2^j \cdot (c_j + [r_j] - 2 \cdot c_j \cdot [r_j]) \bmod 2^k$.

As a natural modification of Protocol 27, we note that H_1 and H_2 are not required if it is certain that H_0 is a semi-honest client. One client is more natural in the setting than three clients. However, we stress that this modified protocol places a stronger assumption on the clients than Protocol 27 does.

The modified protocol is Protocol 28. It requires 1 round and $12k^2 + 12k$ bits in the offline phase and 1 round and $8k$ bits in the online phase as the (amortized) communication cost. Hence, the (amortized) communication cost of Protocol 28 is lower than that of Protocol 27.

5.3.1.4 N-party and H-client Construction with Malicious Corrupted Parties and Clients

---

Protocol 29 $\{[\![r_j]\!]^{\mathsf{B}}, [\![r_j]\!]\}_{j=0}^{n-1} \leftarrow \pi_{\mathsf{rMultiRndGen}}(F, F', \mathcal{H}, \mathsf{seed}, \{\mathsf{vid}_j, \mathsf{vid}_j^{(1)}, \ldots, \mathsf{vid}_j^{(N-1)}\}_{j=0}^{n-1})$

Input: pseudo-random function $F : \{0,1\}^\kappa \times \{0,1\}^\kappa \to \mathbb{Z}_2$ and $F' : \{0,1\}^\kappa \times \{0,1\}^\kappa \to \mathbb{Z}_{2^k}$, collision-resistant hash function $\mathcal{H}$, shared seed by clients seed, unique identifiers $\{\mathsf{vid}_j, \mathsf{vid}_j^{(1)}, \ldots, \mathsf{vid}_j^{(N-1)}\}_{j=0}^{n-1}$ *s.t.* $\mathsf{seed}, \mathsf{vid}_j, \mathsf{vid}_j^{(1)}, \ldots, \mathsf{vid}_j^{(N-1)} \in \{0,1\}^\kappa$ (n is a positive integer)

Output: $\{[\![r_j]\!]^{\mathsf{B}}, [\![r_j]\!]\}_{j=0}^{n-1}$ *s.t.* $r_j \in \mathbb{Z}_2$

1: **for** $j = 0, \ldots, n-1$ **do**
2: Each $H_{i'}$ ($i' = 0, \ldots, H-1$) computes $r_j = F(\mathsf{seed}, \mathsf{vid}_j)$.
3: Each $H_{i'}$ computes $r_{j,1} = F(\mathsf{seed}, \mathsf{vid}_j^{(1)}), \ldots r_{j,N-1} = F(\mathsf{seed}, \mathsf{vid}_j^{(N-1)})$.
4: Each $H_{i'}$ sets $r_{j,0} = r_j \oplus r_{j,1} \oplus \cdots \oplus r_{j,N-1} \mod 2$.
5: Each $H_{i'}$ computes $r'_{j,1} = F'(\mathsf{seed}, \mathsf{vid}_j^{(1)}), \ldots r'_{j,N-1} = F'(\mathsf{seed}, \mathsf{vid}_j^{(N-1)})$.
6: Each $H_{i'}$ sets $r'_{j,0} = r_j - \sum_{\ell=1}^{N-1} r'_{j,\ell} \mod 2^k$.
7: Each $H_{i'}$ sets $[\![r_j]\!]_i^{\mathsf{B}} = (r_{j,i}, \ldots, r_{j,i+2t_{\mathsf{p}}})$ for $i = 0, \ldots, N-1$.
8: Each $H_{i'}$ sets $[\![r_j]\!]_i = (r'_{j,i}, \ldots, r'_{j,i+2t_{\mathsf{p}}})$ for $i = 0, \ldots, N-1$.
9: **end for**
10: $H_0 \ldots, H_{t_{\mathsf{c}}}$ send the set of shares $\{[\![r_j]\!]_i^{\mathsf{B}}, [\![r_j]\!]_i\}_{j=0}^{n-1}$ to each P_i ($i = 0, \ldots, N-1$). $H_{i'''}$ ($i''' = t_{\mathsf{c}}+1, \ldots, 2t_{\mathsf{c}}$) computes the hashed values of $\{[\![r_j]\!]_i^{\mathsf{B}}, [\![r_j]\!]_i\}_{j=0}^{n-1}$, $h_{i''',i}$ ($i''' = t_{\mathsf{c}}+1, \ldots, 2t_{\mathsf{c}}$), and send it to each P_i.
`// 1 round & $(t_{\mathsf{c}}+1)(2t_{\mathsf{p}}+1)nk + (t_{\mathsf{c}}+1)(2t_{\mathsf{p}}+1)n$ bits`
11: Let each $R_{i'',i}$ ($i'' = 0, \ldots, t_{\mathsf{c}}$) be the set of shares $\{[\![r_j]\!]_i^{\mathsf{B}}, [\![r_j]\!]_i\}_{j=0}^{n-1}$ that is sent to P_i from $H_{i''}$. Each P_i computes the hashed value of $R_{i'',i}$, $h_{i'',i}$.
12: Each P_i outputs $R_i \in \{R_{i'',i}\}_{i''=0}^{t_{\mathsf{c}}}$ as the correct shares $\{[\![r_j]\!]_i^{\mathsf{B}}, [\![r_j]\!]_i\}_{j=0}^{n-1}$ if $t_{\mathsf{c}}+1$ or more of the hashed values in $\{h_{i',i}\}_{i'=0}^{H-1}$ match the hashed value of $R_{i',i}$ ($i' = 0, \ldots, H-1$).

Protocol 30 $[\![x]\!] \leftarrow \pi_{\mathsf{rMultiBitComp}}^{\mathcal{F}_{\mathsf{rMultiRndGen}}}(\{[\![x_j]\!]^{\mathsf{B}}\}_{j=0}^{k-1})$

Input: $\{[\![x_j]\!]^{\mathsf{B}}\}_{j=0}^{k-1}$ (where $x_j \in \mathbb{Z}_2$)

Output: $[\![x]\!]$ *s.t.* $x = \sum_{j=0}^{k-1} 2^j \cdot x_j \mod 2^k$

1: (Offline phase)
2: All the parties and clients invoke $\mathcal{F}_{\mathsf{rMultiRndGen}}$ where $n = k$, then get $\{[\![r_j]\!]^{\mathsf{B}}, [\![r_j]\!]\}_{j=0}^{k-1}$.
`// $\pi_{\mathsf{rMultiRndGen}}$ requires 1 round & $(t_{\mathsf{c}}+1)(2t_{\mathsf{p}}+1)k^2 + (t_{\mathsf{c}}+1)(2t_{\mathsf{p}}+1)k$ bits`

FUNCTIONALITY 3 ($\mathcal{F}_{\mathsf{rMultiRndGen}}$ - generating two types of random shares of $(t_{\mathsf{p}}+1, N)$-RSS).

1. $\mathcal{F}_{\mathsf{rMultiRndGen}}$ receives message $(\mathsf{gen},\ n)$ from clients $H_{i'}$ for $i' = 0, \ldots, H-1$.
2. $\mathcal{F}_{\mathsf{rMultiRndGen}}$ generates $\{[\![r_j]\!]^{\mathsf{B}},\ [\![r_j]\!]\}_{j=0}^{n-1}$ randomly, where $r_j \in \{0,1\}$.
3. $\mathcal{F}_{\mathsf{rMultiRndGen}}$ sends $\{[\![r_j]\!]^{\mathsf{B}},\ [\![r_j]\!]\}_{j=0}^{n-1}$ to parties P_i for $i = 0, \ldots, N-1$.

FUNCTIONALITY 4 ($\mathcal{F}_{\mathsf{rMultiBitComp}}$ - converting the binary share vector of $(t_{\mathsf{p}}+1, N)$-RSS on $\mathbb{Z}_2$ to the share of $(t_{\mathsf{p}}+1, N)$-RSS on $\mathbb{Z}_{2^k}$).

1. $\mathcal{F}_{\mathsf{rMultiBitComp}}$ receives shares $\{[\![x_j]\!]^{\mathsf{B}}\}_{j=0}^{k-1}$ from parties P_i for $i = 0, \ldots, N-1$.
2. $\mathcal{F}_{\mathsf{rMultiBitComp}}$ reconstructs x_j and computes $x = \sum_{j=0}^{k-1} 2^j \cdot x_j \bmod 2^k$. Then, $\mathcal{F}_{\mathsf{rMultiBitComp}}$ computes $[\![x]\!]$ and sends it to parties P_i for $i = 0, \ldots, N-1$.

3: (Online phase)
4: Each P_i $(i = 0, \ldots, N-1)$ computes $[\![c_j]\!]_i^{\mathsf{B}} = [\![x_j \oplus r_j]\!]_i^{\mathsf{B}} = [\![x_j]\!]_i^{\mathsf{B}} \oplus [\![r_j]\!]_i^{\mathsf{B}}$ for $j = 0, \ldots, k-1$.
5: Each P_i $(i = 0, \ldots, N-1)$ gets the value $c_j \leftarrow \pi_{\mathsf{mbo}}([\![c_j]\!]^{\mathsf{B}})$ for $j = 0, \ldots, k-1$ in parallel. // 1 round & $N(N-(2t_{\mathsf{p}}+1))(t_{\mathsf{p}}+1)k$ bits
6: $[\![x]\!] = \sum_{j=0}^{k-1} 2^j \cdot (c_j + [\![r_j]\!] - 2 \cdot c_j \cdot [\![r_j]\!]) \bmod 2^k$.

Let H and t_{c} be the number of clients and the corruption in the clients. As a modification of Protocol 27, we propose a modified construction with an arbitrary number of parties and clients, Protocol 30, where $t_{\mathsf{p}}(2t_{\mathsf{p}}+1) < N$ and $2t_{\mathsf{c}}+1 < H$. We assume that all clients have seed. We also assume that all clients do not collude with a party as well as the existing client-aided protocols [29, 84, 101].

Since there are no restrictions on the number of parties and clients, Protocol 30 is available for a wider range of situations than Protocol 27.

The computation strategy of Protocol 30 is almost the same as Protocol 27 except that it relies on $(N-2t_{\mathsf{p}}, N)$-RSS, not $(2,4)$-RSS. The proof strategy is also the same as §27. Therefore, Protocol 30 computes $\mathcal{F}_{\mathsf{rMultiBitComp}}$ with computational security if no client colludes with parties and it holds that $t_{\mathsf{p}}(2t_{\mathsf{p}}+1) < N$ and $2t_{\mathsf{c}}+1 < H$.

5.3.2 Client-aided Secure Hamming Distance Calculation Protocol with Private Robustness Independent of Statistical Parameter

5.3.2.1 Protocol

Protocol 31 $[\sum_{j=0}^{m-1}(x_j \oplus y_j)] \leftarrow \pi_{\mathsf{rHD}}^{\mathcal{F}_{\mathsf{rRndGen}}}(\{[x_j]^{\mathsf{B}}, [y_j]^{\mathsf{B}}\}_{j=0}^{m-1})$

Input: $\{[x_j]^{\mathsf{B}}, [y_j]^{\mathsf{B}}\}_{j=0}^{m-1}$ (where $x_j, y_j \in \mathbb{Z}_2$ and $m < 2^k$)

Output: $[\mathsf{dist}]$ *s.t.* $\mathsf{dist} = \sum_{j=0}^{m-1}(x_j \oplus y_j) \mod 2^k$

1: (Offline phase)
2: P_0, P_1, P_2, P_3, H_0, H_1 and H_2 invoke $\mathcal{F}_{\mathsf{rRndGen}}$ where $n = m$, then get $\{[r_j]^{\mathsf{B}}, [r_j]\}_{j=0}^{m-1}$.
// π_{rRndGen} requires 1 round & $24mk + 24m$ bits
3: (Online phase)
4: Each P_i computes $[z_j]_i^{\mathsf{B}} = [x_j \oplus y_j]_i^{\mathsf{B}} = [x_j]_i^{\mathsf{B}} \oplus [y_j]_i^{\mathsf{B}}$ for $j = 0, \ldots, m-1$.
5: Each P_i computes $[c_j]_i^{\mathsf{B}} = [z_j \oplus r_j]_i^{\mathsf{B}} = [z_j]_i^{\mathsf{B}} \oplus [r_j]_i^{\mathsf{B}}$ for $j = 0, \ldots, m-1$.
6: Each P_i gets value $c_j \leftarrow \pi_{\mathsf{bo}}([c_j]^{\mathsf{B}})$ for $j = 0, \ldots, m-1$ in parallel.
// 1 round & $8m$ bits
7: $[\mathsf{dist}] = \sum_{j=0}^{m-1}(c_j + [r_j] - 2 \cdot c_j \cdot [r_j]) \bmod 2^k$.

FUNCTIONALITY 5 ($\mathcal{F}_{\mathsf{rHD}}$ - computing the share of the Hamming distance between $(x_0, \ldots, x_{m-1})$ and $(y_0, \ldots, y_{m-1})$ from $\{[x_j]^{\mathsf{B}}, [y_j]^{\mathsf{B}}\}_{j=0}^{m-1}$, where $x_j, y_j \in \{0, 1\}$ and $m < 2^k$).

1. $\mathcal{F}_{\mathsf{rHD}}$ receives shares $\{[x_j]^{\mathsf{B}}, [y_j]^{\mathsf{B}}\}_{j=0}^{m-1}$ from parties P_0, P_1, P_2, and P_3.
2. $\mathcal{F}_{\mathsf{rHD}}$ reconstructs $x_j, y_j (j = 0, \ldots, m)$ and computes $\mathsf{dist} = \sum_{j=0}^{m-1}(x_j \oplus y_j) \bmod 2^k$. Then, $\mathcal{F}_{\mathsf{rHD}}$ computes $[\mathsf{dist}]$ and sends it to parties P_0, P_1, P_2 and P_3.

Our secure Hamming distance calculation protocol $\pi_{\mathsf{rHD}}^{\mathcal{F}_{\mathsf{rRndGen}}}$ in Protocol 31 is based on $\pi_{\mathsf{rBitComp}}^{\mathcal{F}_{\mathsf{rRndGen}}}$. In the offline phase, $\mathcal{F}_{\mathsf{rRndGen}}$ (where $n = m$) is invoked in the same way as $\pi_{\mathsf{rBitComp}}^{\mathcal{F}_{\mathsf{rRndGen}}}$ (at Line 2 in Protocol 31). In the online phase, each P_i computes $[z_j]_i^{\mathsf{B}} = [x_j \oplus y_j]_i^{\mathsf{B}}$ (at Line 4). Then, each P_i gets masked value $c_j (\in \{0, 1\}) = z_j \oplus r_j$ $(j = 0, \ldots, m-1)$ by using shares $\{[r_j]^{\mathsf{B}}\}_{j=0}^{m-1}$ and π_{bo} (at Lines 5 and 6 in Protocol 31). Finally, the parties remove mask r_j from c_j by computing $[x_j] = [c_j \oplus r_j] = (c_j - [r_j])^2 = c_j + [r_j] - 2 \cdot c_j \cdot [r_j] \bmod 2^k$. Then, the parties output $[\mathsf{dist}] = [\sum_{j=0}^{m-1} z_j] = \sum_{j=0}^{m-1}(c_j + [r_j] - 2 \cdot c_j \cdot [r_j])$ (at Line 7 in Protocol 31). The security of our protocol is proved in the same way as for $\pi_{\mathsf{rBitComp}}^{\mathcal{F}_{\mathsf{rRndGen}}}$.

5.3.2.2 Application Setting

In biometric authentication services that use iris recognition, a user has the biometric template, the binary vector $\{x_j\}_{j=0}^{m-1}$. The servicer has the registered template, the binary vector $\{y_j\}_{j=0}^{m-1}$. We assume that the authentication of the user is successful if the Hamming distance $\sum_{j=0}^{m-1} x_j \oplus y_j$ is smaller than the decision threshold value d, which the servicer has. Note that the user and servicer do not want to reveal the vector to each other to ensure privacy and prevent information leakage.

We assume that the (honest) user and the (honest) servicer compute $\{[x_j]^{\mathsf{B}}\}_{j=0}^{m-1}$ and $\{[y_j]^{\mathsf{B}}\}_{j=0}^{m-1}$ and send them to the MPC servers (P_0, P_1, P_2 and P_3) running Protocol 31, respectively. Then, all the MPC servers send their share of the Hamming distance $[\mathsf{dist}]$ to

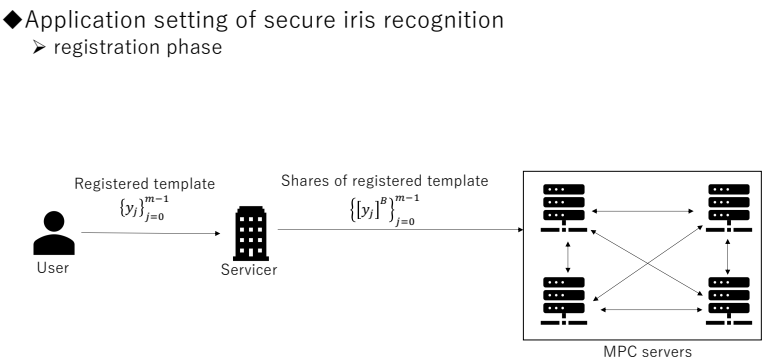

Figure 5.1: Overview of registration phase in secure iris recognition

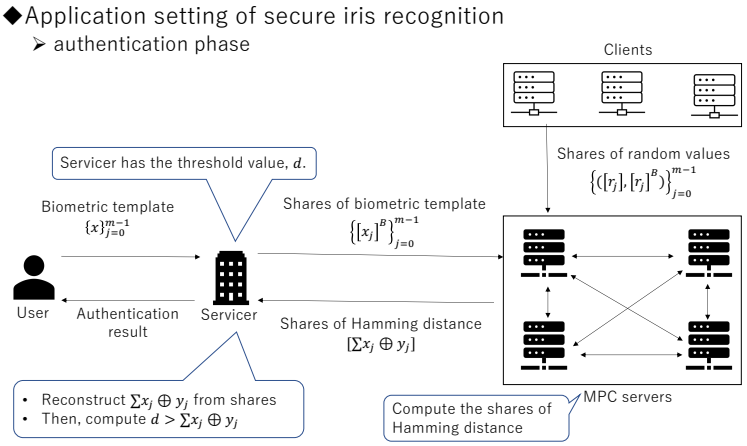

Figure 5.2: Overview of authentication phase in secure iris recognition

the servicer. Then, the servicer reconstructs dist and selects two or more matching values as the correct output. Finally, the servicer checks whether dist is smaller than d and sends the result of the authentication to the user. In this way, our scheme can provide a secure authentication service that is robust against DoS attacks in the standard model.

Fig. 5.1 and 5.2 show the overviews of registration and authentication phases in the biometric authentication service that uses the iris recognition.

5.3.2.3 Modification and Extension of Protocol

Protocol 32 $[\sum_{j=0}^{m-1}(x_j \oplus y_j)] \leftarrow \pi_{\mathsf{rHD5}}(\{[x_j]^{\mathsf{B}}, [y_j]^{\mathsf{B}}\}_{j=0}^{m-1})$

Input: $\{[x_j]^{\mathsf{B}}, [y_j]^{\mathsf{B}}\}_{j=0}^{m-1}$ (where $x_j, y_j \in \mathbb{Z}_2$ and $m < 2^k$)
Output: $[\mathsf{dist}]$ *s.t.* $\mathsf{dist} = \sum_{j=0}^{m-1}(x_j \oplus y_j) \mod 2^k$

1: (Offline phase)
2: H_0 generates $\{[r_j]^{\mathsf{B}}, [r_j]\}_{j=0}^{m-1}$ randomly and distributes it to the parties. `// 1 round &` $12mk + 12m$ `bits`
3: (Online phase)
4: Each P_i $(i = 0, \ldots, 3)$ computes $[z_j]_i^{\mathsf{B}} = [x_j \oplus y_j]_i^{\mathsf{B}} = [x_j]_i^{\mathsf{B}} \oplus [y_j]_i^{\mathsf{B}}$ for $j = 0, \ldots, m-1$.
5: Each P_i computes $[c_j]_i^{\mathsf{B}} = [z_j \oplus r_j]_i^{\mathsf{B}} = [z_j]_i^{\mathsf{B}} \oplus [r_j]_i^{\mathsf{B}}$ for $j = 0, \ldots, m-1$.
6: Each P_i gets value $c_j \leftarrow \pi_{\mathsf{bo}}([c_j]^{\mathsf{B}})$ for $j = 0, \ldots, m-1$ in parallel. `// 1 round &` $8m$ `bits`
7: $[\mathsf{dist}] = \sum_{j=0}^{m-1}(c_j + [r_j] - 2 \cdot c_j \cdot [r_j]) \bmod 2^k$.

Protocol 33 $[\![\sum_{j=0}^{m-1}(x_j \oplus y_j)]\!] \leftarrow \pi_{\mathsf{rMultiHD}}^{\mathcal{F}_{\mathsf{rMultiRndGen}}}(\{[\![x_j]\!]^{\mathsf{B}}, [\![y_j]\!]^{\mathsf{B}}\}_{j=0}^{m-1})$

Input: $\{[\![x_j]\!]^{\mathsf{B}}, [\![y_j]\!]^{\mathsf{B}}\}_{j=0}^{m-1}$ (where $x_j, y_j \in \mathbb{Z}_2$ and $m < 2^k$)
Output: $[\![\mathsf{dist}]\!]$ *s.t.* $\mathsf{dist} = \sum_{j=0}^{m-1}(x_j \oplus y_j) \mod 2^k$

1: (Offline phase)
2: All the parties and clients invoke $\mathcal{F}_{\mathsf{rMultiRndGen}}$ where $n = m$, then get $\{[\![r_j]\!]^{\mathsf{B}}, [\![r_j]\!]\}_{j=0}^{m-1}$. `//` $\pi_{\mathsf{rMultiRndGen}}$ `requires 1 round &` $(t_{\mathsf{c}}+1)(2t_{\mathsf{p}}+1)mk + (t_{\mathsf{c}}+1)(2t_{\mathsf{p}}+1)m$ `bits`
3: (Online phase)
4: Parties compute $[\![z_j]\!]^{\mathsf{B}} = [\![x_j \oplus y_j]\!]^{\mathsf{B}} = [\![x_j]\!]^{\mathsf{B}} \oplus [\![y_j]\!]^{\mathsf{B}}$ for $j = 0, \ldots, m-1$.
5: Parties compute $[\![c_j]\!]^{\mathsf{B}} = [\![z_j \oplus r_j]\!]^{\mathsf{B}} = [\![z_j]\!]^{\mathsf{B}} \oplus [\![r_j]\!]^{\mathsf{B}}$ for $j = 0, \ldots, m-1$.
6: Each party gets value $c_j \leftarrow \pi_{\mathsf{mbo}}([\![c_j]\!]^{\mathsf{B}})$ for $j = 0, \ldots, m-1$ in parallel. `// 1 round &` $N(N - (2t_{\mathsf{p}}+1))(t_{\mathsf{p}}+1)m$ `bits`
7: $[\![\mathsf{dist}]\!] = \sum_{j=0}^{m-1}(c_j + [r_j] - 2 \cdot c_j \cdot [\![r_j]\!]) \bmod 2^k$.

We can modify and extend Protocol 31 by using Protocols 28 and 30. Protocol 32 (i.e., the modified Hamming distance calculation protocol based on Protocol 28) requires only one client. Hence, the financial cost of the service with Protocol 32 is lower than with Protocol 31. In Protocol 33 (i.e., the extended protocol based on Protocol 30), there can be an

arbitrary number of parties and clients. Therefore, Protocol 33 is useful for improving the system redundancy and high availability of services.

5.4 Summary

In this chapter, we proposed the client-aided maliciously secure bit-composition protocol with GOD (private robustness independent of a statistical parameter) in the standard model. Our scheme simultaneously improves the efficiency of computing complex functions and the security. We also proposed the secure Hamming distance protocol with GOD (private robustness independent of a statistical parameter) in the standard model by modifying our bit-composition protocol. Our Hamming distance protocol can help provide a secure iris recognition service that is robust against DoS attacks.

Chapter 6

Conclusion

We proposed the ring-based SS-MPC protocol for the constant-round PDTE [1, 19, 22, 23] and that for the iris authentication with robustness [20, 21].

Our constant-round PDTE protocols with semi-honest security [22, 23] would be a candidate for a high-performance method, even in an environment with high communication latency, as an alternative to the approach using GC and HE. Our constant-round PDTE protocols with malicious security and fairness [1, 19] achieve not only the same level of usefulness as the constant-round PDTE protocols with semi-honest security [22, 23], but also a stronger security notion than those protocols because it is secure for an attacker to deviate from the protocol specification.

Secure iris recognition services based on our Hamming distance calculation protocols with robustness [20, 21] is secure against DoS attacks, making it practical for both service availability and security. This is because our protocols achieve robustness, and abort does not occur.

We leave as a future work the proposal of an efficient mixed protocol framework supporting SS-MPC and GC that achieves private robustness. A mixed protocol framework supporting SS-MPC and GC that achieves traditional robustness [18] has been proposed, but it with private robustness has not yet been proposed. We also as a future work the proposal of an efficient multiparty protocol achieving FaF-security [102]. Loosely speaking, FaF-security is a more formal security definition than private robustness.

Acknowledgements

I would like to express my sincere appreciation to Associate Professor Takashi Nishide for his thoughtful guidance and continuous support. He provided useful comments on my dissertation and some of my research [1, 19–23].

I also thank Mr. Yusaku Maeda, co-author of several studies [22, 23]. He shared with me his knowledge of Demux that helped me construct the OAR protocol.

I am grateful to the members of the dissertation committee, Professor Noboru Kunihiro, Professor Kazumasa Omote, Associate Professor Kazuki Katagishi, and Visiting Associate Professor Masaki Shimaoka.

Finally, I would like to thank the anonymous reviewers and potential readers for their helpful comments on my research. I hope that my research can arouse the interest of even a few potential readers.

Bibliography

[1] Hikaru TSUCHIDA and Takashi NISHIDE. Constant-round fair ss-4pc for private decision tree evaluation. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, advpub:2021DMP0016, 2022.

[2] Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract). In *STOC*, pages 1–10. ACM, 1988.

[3] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game or A completeness theorem for protocols with honest majority. In *STOC*, pages 218–229. ACM, 1987.

[4] Andrew Chi-Chih Yao. How to generate and exchange secrets (extended abstract). In *FOCS*, pages 162–167. IEEE Computer Society, 1986.

[5] Toshinori Araki, Jun Furukawa, Yehuda Lindell, Ariel Nof, and Kazuma Ohara. High-throughput semi-honest secure three-party computation with an honest majority. In *ACM Conference on Computer and Communications Security*, pages 805–817. ACM, 2016.

[6] Toshinori Araki, Assi Barak, Jun Furukawa, Marcel Keller, Kazuma Ohara, and Hikaru Tsuchida. How to choose suitable secure multiparty computation using generalized SPDZ. In *ACM Conference on Computer and Communications Security*, pages 2198–2200. ACM, 2018.

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

[8] Jun Furukawa, Yehuda Lindell, Ariel Nof, and Or Weinstein. High-throughput secure three-party computation for malicious adversaries and an honest majority. In *EUROCRYPT (2)*, volume 10211 of *Lecture Notes in Computer Science*, pages 225–255, 2017.

[9] Toshinori Araki, Assi Barak, Jun Furukawa, Tamar Lichter, Yehuda Lindell, Ariel Nof, Kazuma Ohara, Adi Watzman, and Or Weinstein. Optimized honest-majority

mpc for malicious adversaries - breaking the 1 billion-gate per second barrier. In *IEEE Symposium on Security and Privacy*, pages 843–862. IEEE Computer Society, 2017.

[10] Payman Mohassel and Peter Rindal. Aby^3: A mixed protocol framework for machine learning. In *ACM Conference on Computer and Communications Security*, pages 35–52. ACM, 2018.

[11] Peeter Laud and Alisa Pankova. Bit decomposition protocols in secure multiparty computation. In *WAHC@CCS*, pages 37–48. ACM, 2018.

[12] Harsh Chaudhari, Rahul Rachuri, and Ajith Suresh. Trident: Efficient 4pc framework for privacy preserving machine learning. In *NDSS*. The Internet Society, 2020.

[13] Arpita Patra and Ajith Suresh. BLAZE: blazing fast privacy-preserving machine learning. In *NDSS*. The Internet Society, 2020.

[14] Megha Byali, Harsh Chaudhari, Arpita Patra, and Ajith Suresh. FLASH: fast and robust framework for privacy-preserving machine learning. *Proc. Priv. Enhancing Technol.*, 2020(2):459–480, 2020.

[15] S. Dov Gordon, Samuel Ranellucci, and Xiao Wang. Secure computation with low communication from cross-checking. In *ASIACRYPT (3)*, volume 11274 of *Lecture Notes in Computer Science*, pages 59–85. Springer, 2018.

[16] Anders P. K. Dalskov, Daniel Escudero, and Marcel Keller. Fantastic four: Honest-majority four-party secure computation with malicious security. In *USENIX Security Symposium*, pages 2183–2200. USENIX Association, 2021.

[17] Nishat Koti, Mahak Pancholi, Arpita Patra, and Ajith Suresh. SWIFT: super-fast and robust privacy-preserving machine learning. In *USENIX Security Symposium*, pages 2651–2668. USENIX Association, 2021.

[18] Nishat Koti, Arpita Patra, Rahul Rachuri, and Ajith Suresh. Tetrad: Actively secure 4PC for secure training and inference. Cryptology ePrint Archive, Report 2021/755, 2021. `https://eprint.iacr.org/2021/755` (Accepted to NDSS 2022).

[19] Hikaru Tsuchida and Takashi Nishide. Private decision tree evaluation with constant rounds via (only) fair SS-4PC. In *ACISP*, volume 13083 of *Lecture Notes in Computer Science*, pages 309–329. Springer, 2021.

[20] Hikaru Tsuchida and Takashi Nishide. Client-aided bit-composition protocol with guaranteed output delivery. In *2020 International Symposium on Information Theory and Its Applications (ISITA)*. IEEE, 2020.

[21] Hikaru Tsuchida and Takashi Nishide. Client-aided robust bit-composition protocol with deterministic cheater identification in standard model. *J. Inf. Process.*, 29:515–524, 2021.

[22] Hikaru Tsuchida, Takashi Nishide, and Yusaku Maeda. Private decision tree evaluation with constant rounds via (only) SS-3PC over ring. In *ProvSec*, volume 12505 of *Lecture Notes in Computer Science*, pages 298–317. Springer, 2020.

[23] Hikaru Tsuchida, Takashi Nishide, and Yusaku Maeda. Private decision tree evaluation with constant rounds via (only) SS-3PC over ring and field. *IEICE Trans. Fundam. Electron. Commun. Comput. Sci.*, 105-A(3):214–230, 2022.

[24] David W. Archer, Dan Bogdanov, Liina Kamm, Y. Lindell, Kurt Nielsen, Jakob Illeborg Pagter, Nigel P. Smart, and Rebecca N. Wright. From keys to databases – real-world applications of secure multi-party computation. Cryptology ePrint Archive, Paper 2018/450, 2018. `https://eprint.iacr.org/2018/450`.

[25] Dan Bogdanov. How to securely perform computations on secret-shared data. *Mater's Thesis*, 2007.

[26] Dan Bogdanov, Sven Laur, and Jan Willemson. Sharemind: A framework for fast privacy-preserving computations. In *European Symposium on Research in Computer Security*, pages 192–206. Springer, 2008.

[27] Dan Bogdanov. *Sharemind: programmable secure computations with practical applications*. PhD thesis, University of Tartu, 2013. `https://dspace.ut.ee/handle/10062/29041`.

[28] Peter Bogetoft, Dan Lund Christensen, Ivan Damgård, Martin Geisler, Thomas P. Jakobsen, Mikkel Krøigaard, Janus Dam Nielsen, Jesper Buus Nielsen, Kurt Nielsen, Jakob Pagter, Michael I. Schwartzbach, and Tomas Toft. Secure multiparty computation goes live. In *Financial Cryptography*, volume 5628 of *Lecture Notes in Computer Science*, pages 325–343. Springer, 2009.

[29] Payman Mohassel and Yupeng Zhang. Secureml: A system for scalable privacy-preserving machine learning. In *IEEE Symposium on Security and Privacy*, pages 19–38. IEEE Computer Society, 2017.

[30] Daniel Demmler, Thomas Schneider, and Michael Zohner. ABY - A framework for efficient mixed-protocol secure two-party computation. In *NDSS*. The Internet Society, 2015.

[31] M. Sadegh Riazi, Christian Weinert, Oleksandr Tkachenko, Ebrahim M. Songhori, Thomas Schneider, and Farinaz Koushanfar. Chameleon: A hybrid secure computation framework for machine learning applications. In *AsiaCCS*, pages 707–721. ACM, 2018.

[32] Jian Liu, Mika Juuti, Yao Lu, and N. Asokan. Oblivious neural network predictions via minionn transformations. In *CCS*, pages 619–631. ACM, 2017.

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

[34] Nishanth Chandran, Divya Gupta, Aseem Rastogi, Rahul Sharma, and Shardul Tripathi. Ezpc: Programmable and efficient secure two-party computation for machine learning. In *EuroS&P*, pages 496–511. IEEE, 2019.

[35] Nishant Kumar, Mayank Rathee, Nishanth Chandran, Divya Gupta, Aseem Rastogi, and Rahul Sharma. Cryptflow: Secure tensorflow inference. In *IEEE Symposium on Security and Privacy*, pages 336–353. IEEE, 2020.

[36] Sameer Wagh, Shruti Tople, Fabrice Benhamouda, Eyal Kushilevitz, Prateek Mittal, and Tal Rabin. Falcon: Honest-majority maliciously secure framework for private deep learning. *Proc. Priv. Enhancing Technol.*, 2021(1):188–208, 2021.

[37] Sebastiaan de Hoogh, Berry Schoenmakers, Ping Chen, and Harm op den Akker. Practical secure decision tree learning in a teletreatment application. In *Financial Cryptography*, volume 8437 of *Lecture Notes in Computer Science*, pages 179–194. Springer, 2014.

[38] Yehuda Lindell and Benny Pinkas. Privacy preserving data mining. In *CRYPTO*, volume 1880 of *Lecture Notes in Computer Science*, pages 36–54. Springer, 2000.

[39] Ágnes Kiss, Masoud Naderpour, Jian Liu, N. Asokan, and Thomas Schneider. Sok: Modular and efficient private decision tree evaluation. *PoPETs*, 2019(2):187–208, 2019.

[40] Mark Abspoel, Daniel Escudero, and Nikolaj Volgushev. Secure training of decision trees with continuous attributes. *Proc. Priv. Enhancing Technol.*, 2021(1):167–187, 2021.

[41] Wilko Henecka, Stefan Kögl, Ahmad-Reza Sadeghi, Thomas Schneider, and Immo Wehrenberg. TASTY: tool for automating secure two-party computations. In *ACM Conference on Computer and Communications Security*, pages 451–462. ACM, 2010.

[42] Marina Blanton and Paolo Gasti. Secure and efficient protocols for iris and fingerprint identification. In *ESORICS*, volume 6879 of *Lecture Notes in Computer Science*, pages 190–209. Springer, 2011.

[43] Arpita Patra, Thomas Schneider, Ajith Suresh, and Hossein Yalame. ABY2.0: improved mixed-protocol secure two-party computation. In *USENIX Security Symposium*, pages 2165–2182. USENIX Association, 2021.

[44] Amos Treiber, Dirk Müllmann, Thomas Schneider, and Indra Spiecker genannt Döhmann. Data protection law and multi-party computation: Applications to information exchange between law enforcement agencies. In *WPES@CCS*, pages 69–82. ACM, 2022.

[45] Lukas Helminger and Christian Rechberger. Multi-party computation in the GDPR. *IACR Cryptol. ePrint Arch.*, page 491, 2022.

[46] Regulation (eu) 2016/679 of the european parliament and of the council of 27 april 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing directive 95/46/ec (general data protection regulation) (text with eea relevance). `https://eur-lex.europa.eu/eli/reg/2016/679/oj`.

[47] Donald Beaver, Silvio Micali, and Phillip Rogaway. The round complexity of secure protocols (extended abstract). In *STOC*, pages 503–513. ACM, 1990.

[48] Moni Naor and Benny Pinkas. Efficient oblivious transfer protocols. In *SODA*, pages 448–457. ACM/SIAM, 2001.

[49] William Aiello, Yuval Ishai, and Omer Reingold. Priced oblivious transfer: How to sell digital goods. In *EUROCRYPT*, volume 2045 of *Lecture Notes in Computer Science*, pages 119–135. Springer, 2001.

[50] Octavian Catrina and Sebastiaan de Hoogh. Improved primitives for secure multiparty integer computation. In *SCN*, volume 6280 of *Lecture Notes in Computer Science*, pages 182–199. Springer, 2010.

[51] Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *EUROCRYPT*, volume 1592 of *Lecture Notes in Computer Science*, pages 223–238. Springer, 1999.

[52] Ivan Damgård and Mads Jurik. A generalisation, a simplification and some applications of paillier's probabilistic public-key system. In *Public Key Cryptography*, volume 1992 of *Lecture Notes in Computer Science*, pages 119–136. Springer, 2001.

[53] Shafi Goldwasser and Silvio Micali. Probabilistic encryption and how to play mental poker keeping secret all partial information. In *STOC*, pages 365–377. ACM, 1982.

[54] Taher El Gamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Trans. Inf. Theory*, 31(4):469–472, 1985.

[55] Ronald L. Rivest, Adi Shamir, and Leonard M. Adleman. A method for obtaining digital signatures and public-key cryptosystems (reprint). *Commun. ACM*, 26(1):96–99, 1983.

[56] Craig Gentry. *A fully homomorphic encryption scheme*. PhD thesis, Stanford University, 2009. `crypto.stanford.edu/craig`.

[57] Justin Brickell, Donald E. Porter, Vitaly Shmatikov, and Emmett Witchel. Privacy-preserving remote diagnostics. In *ACM Conference on Computer and Communications Security*, pages 498–507. ACM, 2007.

[58] Mauro Barni, Pierluigi Failla, Vladimir Kolesnikov, Riccardo Lazzeretti, Ahmad-Reza Sadeghi, and Thomas Schneider. Secure evaluation of private linear branching programs with medical applications. In *ESORICS*, volume 5789 of *Lecture Notes in Computer Science*, pages 424–439. Springer, 2009.

[59] Martine De Cock, Rafael Dowsley, Caleb Horst, Raj S. Katti, Anderson C. A. Nascimento, Wing-Sea Poon, and Stacey Truex. Efficient and private scoring of decision trees, support vector machines and logistic regression models based on precomputation. *IEEE Trans. Dependable Secur. Comput.*, 16(2):217–230, 2019.

[60] Marina Blanton, Ah Reum Kang, and Chen Yuan. Improved building blocks for secure multi-party computation based on secret sharing with honest majority. In *ACNS (1)*, volume 12146 of *Lecture Notes in Computer Science*, pages 377–397. Springer, 2020.

[61] Peeter Laud. A private lookup protocol with low online complexity for secure multiparty computation. In *ICICS*, volume 8958 of *Lecture Notes in Computer Science*, pages 143–157. Springer, 2014.

[62] Koji Chida, Koki Hamada, Dai Ikarashi, Ryo Kikuchi, Naoto Kiribuchi, and Benny Pinkas. An efficient secure three-party sorting protocol with an honest majority. *IACR Cryptol. ePrint Arch.*, 2019:695, 2019.

[63] Hospital discharge data use agreement. `https://www.dshs.texas.gov/THCIC/Hospitals/Download.shtm`.

[64] Acquire valued shoppers challenge — kaggle. `https://www.kaggle.com/c/acquire-valued-shoppers-challenge/data`.

[65] Vivek Kumar Singh, Burcin Bozkaya, and Alex Pentland. Money walks: implicit mobility behavior and financial well-being. *PloS one*, 10(8):e0136628, 2015.

[66] J. Ross Quinlan. Induction of decision trees. *Machine learning*, 1(1):81–106, 1986.

[67] Sven Laur, Jan Willemson, and Bingsheng Zhang. Round-efficient oblivious database manipulation. In *ISC*, volume 7001 of *Lecture Notes in Computer Science*, pages 262–277. Springer, 2011.

[68] Koki Hamada, Ryo Kikuchi, Dai Ikarashi, Koji Chida, and Katsumi Takahashi. Practically efficient multi-party sorting protocols from comparison sort algorithms. In *ICISC*, volume 7839 of *Lecture Notes in Computer Science*, pages 202–216. Springer, 2012.

[69] Koki Hamada, Dai Ikarashi, Koji Chida, and Katsumi Takahashi. Oblivious radix sort: An efficient sorting algorithm for practical secure multi-party computation. *IACR Cryptol. ePrint Arch.*, 2014:121, 2014.

[70] Keitaro Hiwatashi, Ken Ogura, Satsuya Ohata, and Koji Nuida. Accelerating secure (2+1)-party computation by insecure but efficient building blocks. In *AsiaCCS*, pages 616–627. ACM, 2021.

[71] Harsh Chaudhari, Ashish Choudhury, Arpita Patra, and Ajith Suresh. ASTRA: high throughput 3pc over rings with application to secure prediction. In *CCSW@CCS*, pages 81–92. ACM, 2019.

[72] Marcel Keller and Peter Scholl. Efficient, oblivious data structures for MPC. In *ASIACRYPT (2)*, volume 8874 of *Lecture Notes in Computer Science*, pages 506–525. Springer, 2014.

[73] Peeter Laud. Parallel oblivious array access for secure multiparty computation and privacy-preserving minimum spanning trees. *PoPETs*, 2015(2):188–205, 2015.

[74] John Launchbury, Iavor S. Diatchki, Thomas DuBuisson, and Andy Adams-Moran. Efficient lookup-table protocol in secure multiparty computation. In *ICFP*, pages 189–200. ACM, 2012.

[75] Jack Doerner and Abhi Shelat. Scaling ORAM for secure computation. In *ACM Conference on Computer and Communications Security*, pages 523–535. ACM, 2017.

[76] Xiao Shaun Wang, Yan Huang, T.-H. Hubert Chan, Abhi Shelat, and Elaine Shi. SCORAM: oblivious RAM for secure computation. In *ACM Conference on Computer and Communications Security*, pages 191–202. ACM, 2014.

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

[78] Xiao Wang, T.-H. Hubert Chan, and Elaine Shi. Circuit ORAM: on tightness of the goldreich-ostrovsky lower bound. *IACR Cryptol. ePrint Arch.*, 2014:672, 2014.

[79] Sky Faber, Stanislaw Jarecki, Sotirios Kentros, and Boyang Wei. Three-party ORAM for secure computation. In *ASIACRYPT (1)*, volume 9452 of *Lecture Notes in Computer Science*, pages 360–385. Springer, 2015.

[80] Stanislaw Jarecki and Boyang Wei. 3pc ORAM with low latency, low bandwidth, and fast batch retrieval. In *ACNS*, volume 10892 of *Lecture Notes in Computer Science*, pages 360–378. Springer, 2018.

[81] Daniel Escudero, Satrajit Ghosh, Marcel Keller, Rahul Rachuri, and Peter Scholl. Improved primitives for MPC over mixed arithmetic-binary circuits. In *CRYPTO (2)*, volume 12171 of *Lecture Notes in Computer Science*, pages 823–852. Springer, 2020.

[82] Anders P. K. Dalskov, Daniel Escudero, and Marcel Keller. Secure evaluation of quantized neural networks. *Proc. Priv. Enhancing Technol.*, 2020(4):355–375, 2020.

[83] Ivan Damgård, Daniel Escudero, Tore Kasper Frederiksen, Marcel Keller, Peter Scholl, and Nikolaj Volgushev. New primitives for actively-secure MPC over rings with applications to private machine learning. In *IEEE Symposium on Security and Privacy*, pages 1102–1120. IEEE, 2019.

[84] Satsuya Ohata and Koji Nuida. Communication-efficient (client-aided) secure two-party protocols and its application. In *Financial Cryptography*, volume 12059 of *Lecture Notes in Computer Science*, pages 369–385. Springer, 2020.

[85] David J. Wu, Tony Feng, Michael Naehrig, and Kristin E. Lauter. Privately evaluating decision trees and random forests. *PoPETs*, 2016(4):335–355, 2016.

[86] Raymond K. H. Tai, Jack P. K. Ma, Yongjun Zhao, and Sherman S. M. Chow. Privacy-preserving decision trees evaluation via linear functions. In *ESORICS (2)*, volume 10493 of *Lecture Notes in Computer Science*, pages 494–512. Springer, 2017.

[87] Atsunori Ichikawa, Wakaha Ogata, Koki Hamada, and Ryo Kikuchi. Efficient secure multi-party protocols for decision tree classification. In *ACISP*, volume 11547 of *Lecture Notes in Computer Science*, pages 362–380. Springer, 2019.

[88] Anselme Tueno, Florian Kerschbaum, and Stefan Katzenbeisser. Private evaluation of decision trees using sublinear cost. *PoPETs*, 2019(1):266–286, 2019.

[89] Payman Mohassel and Seyed Saeed Sadeghian. How to hide circuits in MPC an efficient framework for private function evaluation. In *EUROCRYPT*, volume 7881 of *Lecture Notes in Computer Science*, pages 557–574. Springer, 2013.

[90] Yan Huang, David Evans, and Jonathan Katz. Private set intersection: Are garbled circuits better than custom protocols? In *NDSS*. The Internet Society, 2012.

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

[92] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *FOCS*, pages 136–145. IEEE Computer Society, 2001.

[93] Rosario Gennaro, Michael O. Rabin, and Tal Rabin. Simplified VSS and fast-track multiparty computations with applications to threshold cryptography. In *PODC*, pages 101–111. ACM, 1998.

[94] Jun Furukawa and Yehuda Lindell. Two-thirds honest-majority MPC for malicious adversaries at almost the cost of semi-honest. In *CCS*, pages 1557–1571. ACM, 2019.

[95] Eyal Kushilevitz, Yehuda Lindell, and Tal Rabin. Information-theoretically secure protocols and security under composition. *SIAM J. Comput.*, 39(5):2090–2112, 2010.

[96] Abdelrahaman Aly, Emmanuela Orsini, Dragos Rotaru, Nigel P. Smart, and Tim Wood. Zaphod: Efficiently combining LSSS and garbled circuits in SCALE. In *WAHC@CCS*, pages 33–44. ACM, 2019.

[97] Donald Beaver. Commodity-based cryptography (extended abstract). In *STOC*, pages 446–455. ACM, 1997.

[98] Megha Byali, Carmit Hazay, Arpita Patra, and Swati Singla. Fast actively secure five-party computation with security beyond abort. In *ACM Conference on Computer and Communications Security*, pages 1573–1590. ACM, 2019.

[99] Megha Byali, Arun Joseph, Arpita Patra, and Divya Ravi. Fast secure computation for small population over the internet. In *ACM Conference on Computer and Communications Security*, pages 677–694. ACM, 2018.

[100] Nishanth Chandran, Juan A. Garay, Payman Mohassel, and Satyanarayana Vusirikala. Efficient, constant-round and actively secure MPC: beyond the three-party case. In *CCS*, pages 277–294. ACM, 2017.

[101] Hiraku Morita, Nuttapong Attrapadung, Tadanori Teruya, Satsuya Ohata, Koji Nuida, and Goichiro Hanaoka. Constant-round client-aided secure comparison protocol. In *ESORICS (2)*, volume 11099 of *Lecture Notes in Computer Science*, pages 395–415. Springer, 2018.

[102] Bar Alon, Eran Omri, and Anat Paskin-Cherniavsky. MPC with friends and foes. In *CRYPTO (2)*, volume 12171 of *Lecture Notes in Computer Science*, pages 677–706. Springer, 2020.