Delegated Private Matching for Compute

Dimitris Mouris[1], Daniel Masny[2], Ni Trieu[3], Shubho Sengupta[2], Prasad Buddhavarapu[2], and Benjamin Case[2]

[1] University of Delaware
[2] Meta Inc.
[3] Arizona State University

Abstract. Private matching for compute (PMC) establishes a match between two datasets owned by mutually distrusted parties (C and P) and allows the parties to input more data for the matched records for arbitrary downstream secure computation without rerunning the private matching component. The state-of-the-art PMC protocols only support two parties and assume that both parties can participate in computationally intensive secure computation. We observe that such operational overhead limits the adoption of these protocols to solely powerful entities as small data owners or devices with minimal computing power will not be able to participate.

We introduce two protocols to *delegate* PMC from party P to untrusted cloud servers, called *delegates*, allowing multiple smaller P parties to provide inputs containing identifiers and associated values. Our *Delegated Private Matching for Compute* protocols, called DPMC and D$_s$PMC, establish a join between the datasets of party C and multiple delegators P based on multiple identifiers and compute secret shares of associated values for the identifiers that the parties have in common. We introduce a rerandomizable encrypted oblivious pseudorandom function (OPRF) primitive, called EO, which allows two parties to encrypt, mask, and shuffle their data. Note that EO may be of independent interest. Our D$_s$PMC protocol limits the leakages of DPMC by combining our EO scheme and secure three-party shuffling. Finally, our implementation demonstrates the efficiency of our constructions by outperforming related works by approximately 10× for the total protocol execution and by at least 20× for the computation on the delegators.

Keywords: Oblivious pseudorandom function · private identity matching · private record linkage · secure multiparty computation

1 Introduction

Cloud computing has become a prominent solution for storage and analytics since it enables clients to outsource their data and not have to worry about scalability, data availability, and most importantly maintaining their own infrastructure. Gathering data from multiple input providers and computing statistics over all of their data enables a plethora of useful applications such as gathering real-time location data and notifying users of possible exposure to highly infectious diseases [25,53]. In certain applications, linking client data to proprietary information owned by multiple larger entities may unlock unique insights that are otherwise not possible. Users should not have to trust that the cloud servers will not store their sensitive personal data for use for purposes other than it's intended. The problem of computing meaningful analytics across multiple input parties while preserving user privacy from cloud server providers becomes significantly more challenging.

Secure multi-party computation (MPC) offers prominent cryptographic solutions for jointly computing on private data from multiple input providers [55,37,27]. Although general-purpose MPC frameworks [6,54,21,29,32] enable running arbitrary computations over private data (e.g., medical data analytics [26]), they generally incur significant performance overheads compared to solutions that are tailored to one application (e.g., machine learning [33] and statistics [15,42,7,41,20]). Similarly, specialized private set intersection (PSI) protocols [49,34,14,46,47,24,13,51,50,12] introduce significantly more efficient solutions than generic MPC but focus solely on private matching and disregard associated metadata.

A few recent protocols inspired by [38] that are based on the hardness of Decisional Diffie–Hellman (DDH) have attempted to securely link private records and allow general-purpose secure computation on the common data. More specifically, private matching for compute (PMC) [10,8,43] from Meta, private set intersection

(PSI) [4] from Apple, and private join and compute (PJC) [31,36] from Google enable computing intersections and unions between two parties while protecting the privacy of the underlying users. After the private linkage is computed, these protocols enable downstream secure computation based on the matched records. Unfortunately, prior works only focus on two parties and require both of them to actively participate in the private matching, which restricts the adoption of these protocols to solely powerful entities as non-crypto-savvy data owners or devices with minimal computing power will not be able to engage in secure computation protocols.

We motivate our work by focusing on the example of *ad attribution*, a crucial business application for tracking the effectiveness of online advertising campaigns and increasing revenue generated by ads. An ad conversion refers to the situation where a user interacts with an online ad for a particular product on the ad publisher's website (which we call party C) and then goes on to make a purchase on the advertiser's website (which we call party P_t for $t \in \{1, \ldots, T\}$). Of course, a large ad publisher may host ads from multiple advertisers, each of which would like to know how their ad campaigns are performing and which purchases can be attributed to their online ads. However, the data required to compute these statistics are split across multiple parties: the ad publisher knows the users who have seen a particular ad, and each advertiser knows who made a purchase and what they spent.

Table 1. Comparisons with related works in terms of functionality, number of parties, threat model, and multi-key support.

Protocol	Functionality	Input Parties		Computing Parties		Delegated	Multi-key
		No.	Model	No.	Model		
PMC [10]	Join (Union)	2	Semi-honest	Same as Input Parties		✗	✗
MK-PMC [8]	Join (Union)	2	Semi-honest	Same as Input Parties		✗	✓
PS^{3}I [10]	Join (Union) & Secret Share Associated Data	2	Semi-honest	Same as Input Parties		✗	✗
PJC [31]	PSI-Sum (Sum Associated Data in Intersection)	2	Semi-honest	Same as Input Parties		✗	✗
Circuit-PSI [48,47,50,12]	PSI & Secret Share Associated Data	2	Semi-honest	Same as Input Parties		✗	✗
Catalic [25]	PSI-CA (Count Items in Intersection)	2	Semi-honest	≥ 2	Semi-honest	✓	✗
DB Joins [40]	Join and Select Statements	2	Semi-honest	3	Semi-honest	✓	✗
DPMC	Join (Left-Join) & Secret Share Associated Data	$T+1$	Semi-honest	2	Semi-honest	✓	✓
D$_s$PMC	Join (Left-Join) & Secret Share Associated Data	$T+1$	Semi-honest	3	Semi-honest	✓	✓

Additionally, in the case of multiple advertisers, the secure ad attribution protocol needs to be repeated for each advertiser, which can be both inefficient and not allow for elaborate statistics. To address this, we propose a *delegated setting* where all advertisers securely delegate the computation to a delegate party, which computes the ad attribution securely with the ad publisher. This approach reduces the computational burden from individual advertisers and also allows for computing cross-advertiser connections, which leads to more advanced applications such as personalization. Users that appear in multiple advertiser datasets are combined into the same row of the final join instead of having multiple joins between the ad publisher and each advertiser. Now the ad publisher and the delegate party can run a secure downstream computation over all advertisers' data. In a real-world instantiation, the ad publisher (e.g., Google, Meta) can collaborate with non-profits (e.g., ISRG) or other companies (e.g., Cloudflare, Mozilla) and allow advertisers to delegate their computation.

To improve match rates, our work considers multi-key matching [8], which involves matching users across multiple identifiers such as email, phone number, and other personal information. By using multiple keys, we can achieve a more accurate match and increase the likelihood of finding a connection between the user who saw the ad and the user who made the purchase. As shown in Fig. 1, User A may be indexed by both an email address and a phone number in party C, whereas, in other data owners, the combination of identifiers may differ (e.g., name, email). In Section 5, we demonstrate the applicability of our work in several domains such as private ad attribution, privacy-preserving analytics, and private machine learning. Note that our approach is not limited to these specific applications and can be extended to other use cases as well.

We ask the following motivating question:

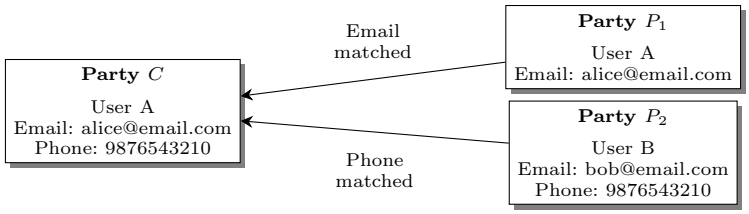


Fig. 1. Many to many connections. P_1 and P_2 have different views about User A and B, whereas Party C has listed them as one user. Notably, P_1 and P_2 might also be the same party.

How can we design delegated private record linkage protocols that allow multiple entities to outsource their records and perform secure computation on the associated metadata of the matched records?

1.1 Previous Works

We now discuss relevant works for private record linkage protocols that allow for computing over associated data. A comparison of our protocols with related works can be found in Table 1.

Private Matching for Compute (PMC) introduced DDH-based constructions for private matching that compute a union of two datasets held by mutually distrusting parties C and P without revealing which items belong to the intersection [10,8]. After the matching phase, both parties can input associated data for each row in the union and engage in a downstream secure computation. The core idea of PMC is to have each party first hash their records and then exponentiate them to random secret scalars. After exchanging the hashed and exponentiated records, each party exponentiates the other party's records to their secret scalar and they both arrive at the same random identifiers.

Multi-key PMC [8] assumes that each record may have multiple identifiers (as shown in Fig. 1) and leverages a ranked deterministic join logic that collapses many-to-many connections and achieves a one-to-one mapping. The idea is that each identifier has a predefined weight and the matching is first performed on all the records based on the first identifier (as the single-key PMC) before continuing to the other identifiers. PMC leaks the intersection size to the two parties and in case of multiple keys per row, they learn the bipartite graph of matches up to an isomorphism. Additionally, PMC supports computing on the match between two datasets and requires both parties to actively participate in both the matching and the downstream secure computation, which significantly limits the adoption in real-world applications. Contrary to PMC, our work allows matching between any number of parties and shifts the cost away from the parties by delegating the computation to a server.

Private secret-shared set intersection (PS^3I) [10] is a natural extension of PMC that allows the two parties to input associated data to the matching protocol. Instead of learning a mapping to original inputs, the two parties only learn additive secret shares of those records which they can feed into any general-purpose MPC framework. PS^3I is realized using Paillier additive homomorphic encryption (HE) scheme [45] and incurs significant performance overheads. Additionally, PS^3I only works between two parties and requires both parties to be online for the whole protocol execution.

Private Join and Compute (PJC) [31,36] computes the intersection between two datasets and aggregates the associated data for all the rows in the intersection using additive HE. Contrary to our work that computes secret shares for all the associated data, PJC only allows computing a sum of the data in the intersection. Furthermore, as with all the previous related works, PJC only supports two parties whereas our protocols scale to multiple parties.

Mohassel et al. [40] utilize cuckoo hash tables and perform SQL-like queries over two secret shared databases in the honest majority three-party setting. Both the input and output tables are secret shared between the computing parties. Because the cuckoo hash tables do not support duplicates, [40] has leakages in the presence of non-unique identifiers. Additionally, the join protocols focus on two parties and in order to compute joins between multiple parties the protocol has to be iterated multiple times. Each party's database has to be joined with the output of the previous join or they can be combined in a binary-tree-like structure. Contrary, our delegated protocols are designed to support multiple delegators and do not have to be repeated for each input party.

Circuit-PSI relies on oblivious Pseudorandom Functions (OPRF) for computing PSI between two parties and then computing a function over the common data [48,47,50,12]. Both parties learn secret shares of 1 or 0, representing record presence in the intersection. These shares are used to input associated data as both parties actively participate in the protocol. These works focus on the two-party setting and it is not clear how to extend them to the delegated setting where multiple parties outsource their data for matching along with encrypted associated data to a helper party. One could imagine that the input parties compute hash tables but then they would need to privately combine these tables which goes beyond what has been studied in the literature. On the other hand, Catalic [25] uses OPRFs between two parties but allows one party to delegate its computation to a powerful server. All the aforementioned works allow two-party matching which is solely based on a single key, while our work supports matching based on datasets of multiple parties where each can have multiple keys (e.g., name, email). Finally, our protocols enable multiple input parties to delegate their computation and then go offline instead of requiring them to participate in expensive protocols.

Miao et al. [39] introduced a shuffled distributed OPRF (DOPRF) for computing PSI between two parties and the shuffling is performed in the clear by one of the parties. Similarly, the authors of [5] propose a DDH-based PRF combined with ElGamal encryption that allows for shuffling by one of the parties. ScrambleDB [35] introduces a three-party OPRF, which evaluates the OPRF on encrypted inputs and also uses re-randomizable encryption to break up the relation of inputs and outputs. As in previous works, ScrambleDB performs shuffling in the clear when data is uploaded, whereas our protocols perform secure shuffling under MPC. We introduce an encrypted oblivious pseudorandom function (OPRF) primitive, called EO, which allows two parties to encrypt, mask, and shuffle their data. While [39,5] seem to be similar to our EO primitive, we emphasize that they are quite different. First of all, our EO primitive performs shuffling under MPC for security contrary to their shuffled protocol. Additionally, although EO could be instantiated with a combination of ElGamal and DH-OPRF, *EO is an abstraction* that can fit many possible instantiations (e.g., from codes, lattices, isogenies). Finally, in this work, we are in the delegated setting to allow multiple parties to outsource the private matching and go offline instead of solely focusing on the two-party setting.

1.2 Our Contributions

Delegated Protocols. We propose a new family of *Delegated Private Matching for Compute* protocols, called DPMC and D_sPMC, that build upon PMC [10,8] and lift the burden of engaging in secure computation from parties with less computational power. Our protocols rely on a powerful server (which we call party C) and on a *delegate* node (which we refer to as party D) to perform private record linkage between the records of C and input parties, which we call *delegators* or parties P. Contrary to previous works that focus on linking data only between two parties, our work enables linkage between C and multiple delegators (P_1 to P_T) and aims to make the computation in the delegators lightweight to foster wide-scale adoption. Parties C and D engage in a two-party computation to compute a private left join of party's C and all the delegators' data.[4] C's input is a multi-key dataset where each row contains multiple identifiers (i.e., keys) that can be matched. The delegators' inputs are multi-key datasets with associated data, which comprise both identifiers and metadata that can be in any form (e.g., numbers, strings). C learns a mapping from its users to the left join but does not learn which of its users have been matched. For each row in the left join, both C and D receive secret shares that correspond to the delegators' associated data if that row maps to one of the delegators' identifiers or a share of `NULL` (i.e., zero), otherwise.

Left Join. Our motivation for performing left join compared to a union or an intersection is that party C learns a mapping from all their users into the join, which allows them to input additional associated metadata without re-executing the matching protocol and without learning which users matched or not. These data can either be labels (in the clear) that could be used to filter the secret shared values (e.g., in a `GROUP BY` fashion), or they can be additional secret shares for the downstream MPC computation. After the matching process and the secret shares have been established, parties C and D only need to know the relative order of their shares, which can then be used for any downstream secure computation such as privacy-preserving analytics and machine learning. Our goal in this work is to create efficient protocols that can be realized in real-world applications for private left join and allow the delegators to outsource the computation to delegates.

[4] Our core protocol computes the left join between party's C data and all delegators' data. We show in Appendix E how to modify it to compute the inner join.

Threat Model. We assume *semi-honest* security, which we prove in the Appendices. Party C follows the protocol specification but tries to exfiltrate information about the delegators' data. Similarly, the delegate D tries to exfiltrate information about all parties' data. Finally, delegators are semi-honest and outsource their real data.

Multi-key Datasets. To increase matching rates, we adopt multi-key datasets [8] and consider matching between records on more than one identifier (i.e., key) that inherently generates many-to-many connections. We use a ranking-based technique to collapse multiple connections into one-to-many (C-to-P) connections. Although we do not claim this contribution, it is an important feature that increases match rates and complicates the protocols; note that it is not straightforward to add this to related works. We operate over unique keys (e.g., email) to avoid inference attacks [44].

Rerandomizable Encrypted OPRF (EO). We introduce a concept called EO that captures the tasks of encrypting identifiers, shuffling ciphertexts, homomorphically evaluating a PRF on the ciphertexts, and decrypting a homomorphically evaluated ciphertext to the PRF output with the identifier as input. More specifically, one party can evaluate the OPRF both on clear data (e.g., x) and encrypted data (e.g., $\mathsf{Enc}(x)$), and then delegate the matching to another party that can decrypt the evaluated OPRF of the encrypted data and get $\mathsf{Dec}(\mathsf{PRF}(\mathsf{Enc}(x))) = \mathsf{PRF}(x)$ (keys are omitted here, see Section 3.3). Further, EO allows shuffling encrypted inputs such that the third party cannot correlate PRF outputs and the initially received ciphertexts. Notice that EO is more powerful than an OPRF since it allows encrypting inputs for the PRF and sends the ciphertexts to a third party (i.e., delegate the evaluation) whereas an OPRF asks that the input provider directly interacts with the PRF evaluator. This allows us to reduce the leakage to the third party (i.e., Party D). Furthermore, the PRF evaluation can be distributed between Party C, who owns the key and homomorphically evaluates the PRF, and Party D, who decrypts the homomorphically evaluated ciphertext and obtains the PRF output.

In Section 3.3, we define the EO primitive and we provide an instantiation based on DDH and ElGamal in Appendix B. Note that EO is an abstraction and can fit various instantiations (possibly from codes, lattices, etc.). Finally, the EO construction might be of independent interest and can facilitate other protocols as well.

$\mathsf{D}_s\mathsf{PMC}$ Protocol. We use our EO primitive to extend DPMC to $\mathsf{D}_s\mathsf{PMC}$, a protocol that uses two delegates (party D and a shuffler S). $\mathsf{D}_s\mathsf{PMC}$ performs an honest majority shuffling protocol between parties C, D, and S and achieves stronger security guarantees in the case of a corruption of Party D and multiple delegators.

Applications. We envision multiple applications that may leverage our delegated setup of merging multiple private datasets and securely computing analytics on metadata. A healthcare provider holding patient records may gain critical insights such as calculating the risk of a health condition by merging with data stored on individual smart devices or other healthcare providers, without needing to access identifiable user data. An ad publisher holding user-provided information may be able to measure advertising efficacy and offer personalized ads by merging with data held by millions of businesses while still preserving user privacy.

Our contributions are summarized as follows:

- We introduce a novel DPMC protocol for securely computing left join between multiple distrusting parties.
- Design of a new rerandomizable encrypted OPRF (EO) primitive that enables encrypting inputs, shuffling ciphertexts, homomorphically evaluating a PRF on ciphertexts, and decrypting ciphertexts to PRF outputs. EO is of independent interest.
- We combine EO and secure three-party shuffling to extend DPMC to $\mathsf{D}_s\mathsf{PMC}$, a protocol that reduces DPMC's leakage.
- We detail applications in online advertising such as privacy-preserving ad attribution, analytics, and personalization.

2 Preliminaries

2.1 Notation

We denote the computational security parameter by κ. We use $[m]$ to refer to the set $\{1, \ldots, m\}$. We denote the concatenation and exclusive OR ($\mathtt{XOR}$) of two-bit strings x and y by $x \,\|\, y$ and $x \oplus y$, respectively. We use

$r \xleftarrow{R} \mathbb{R}$ to refer to a randomly chosen element r from set $\mathbb{R}$. We use ppt to denote probabilistic polynomial time. We use $\{\}$ for unordered and $()$ for ordered sets.

2.2 Definitions

Definition 1 (Multi-Key Key-Value Store). *A multi-key key-value store* KV *is a set of key sets* c_i*, i.e.,* $\mathsf{KV} \coloneqq \{\mathsf{c}_i\}_{i\in[m]}$*. Each set* c_i *contains* m_i *keys, i.e.,* $\mathsf{c}_i \coloneqq \{\mathsf{c}_{i,j}\}_{j\in[m_i]}$*. When the key set is ordered, we denote it with* $\mathsf{c}_i \coloneqq (\mathsf{c}_{i,j})_{j\in[m_i]}$*. Further,* KV *might contain* m *values* v_i *for* $i \in [m]$*, one associated with each key set. In this case, we denote the key sets as* $\mathsf{KV} \coloneqq \{\mathsf{p}_i, \mathsf{v}_i\}_{i\in[m]} = \{\{\mathsf{p}_{i,j}\}_{j\in[m_i]}, \mathsf{v}_i\}_{i\in[m]}$*. Note, we use* p_i *instead of* c_i *when the set includes associated data* v_i*. Furthermore, each key* $\mathsf{c}_{i,j}$ *in a set* KV *is unique, i.e., there does not exist an* $(i', j') \neq (i, j)$ *s.t.* $\mathsf{c}_{i',j'} = \mathsf{c}_{i,j}$*.*

Informally, a multi-key left join with associate data between KV_C and KV_P results in a set of values with as many rows as the set on the left (i.e., KV_C in our case) and the associated values of KV_P for the rows that matched and zero, otherwise.

Definition 2 (Multi-Key Left Join With Associated Data Between Two Key-Value Stores). *Let* $\mathsf{KV}_C \coloneqq \{\mathsf{c}_i\}_{i\in[m_C]}$ *be a multi-key set of party* C *that contains ordered key sets, i.e.,* $\mathsf{c}_i \coloneqq (\mathsf{c}_{i,j})_{j\in[m_{C,i}]}$*. Let* $\mathsf{KV}_P \coloneqq \{\mathsf{p}_i, \mathsf{v}_i\}_{i\in[m_P]}$ *be a multi-key key-value set of* P *that contains both key sets, i.e.,* $\mathsf{p}_i \coloneqq \{\mathsf{p}_{i,j}\}_{j\in[m_{P,i}]}$ *and associated values* v_i*. The left join between* KV_C *and* KV_P *is defined by* $\mathsf{KV}_C \bowtie \mathsf{KV}_P \coloneqq (\hat{\mathsf{v}}_i)_{i\in[m_C]}$*, where* $\hat{\mathsf{v}}_i \coloneqq \mathsf{v}_{i'}$ *s.t.* j_i *is the smallest element in* $[m_{C,i}]$ *for which there exists an* $i' \in [m_P]$ *and* $j_{i'} \in [m_{P,i'}]$ *with* $\mathsf{c}_{i,j_i} = \mathsf{p}_{i',j_{i'}}$*. If there does not exist such an* j_i*,* i' *and* $j_{i'}$*, we define* $\hat{\mathsf{v}}_i \coloneqq 0$*.*

With multiple delegators, we extend Def. 2 as follows.

Definition 3 (Multi-Key Left Join With Associated Data Between $T+1$ Key-Value Stores). *Let for all* $t \in [T]$*,* $\mathsf{KV}_t \coloneqq \{\mathsf{p}_{t,i}, \mathsf{v}_{t,i}\}_{i\in[m_t]}$ *be a multi-key key-value set of party* P_t *that contains both key sets, i.e.,* $\mathsf{p}_{t,i} \coloneqq \{\mathsf{p}_{t,i,j}\}_{j\in[m_{t,i}]}$*, and values, i.e.,* $\mathsf{v}_{t,i}$*. Also, let* $\mathsf{KV}_C \coloneqq \{\mathsf{c}_i\}_{i\in[m_C]}$ *be a multi-key set of party* C *that contains only ordered key sets, i.e.,* $\mathsf{c}_i \coloneqq \{\mathsf{c}_{i,j}\}_{j\in[m_{C,i}]}$*. The left join between* KV_C *and* $\{\mathsf{KV}_t\}_{t\in[T]}$ *is defined as:*

$$\mathsf{KV}_C \bowtie \{\mathsf{KV}_1, \ldots, \mathsf{KV}_T\} \coloneqq (\pi_i(\hat{\mathsf{v}}_{i,1}, \ldots, \hat{\mathsf{v}}_{i,T}))_{i\in[m_C]},$$

where for each $t \in [T]$ *and* $i \in [m_C]$*,* $\hat{\mathsf{v}}_{i,t}$ *is defined as follows. Let for each* $j \in [m_{C,i}]$*,* $\mathsf{S}_{i,j,t} \coloneqq \{i' \in [m_t] \mid \exists j' \in [m_{t,i'}]$ *s.t.* $c_{i,j} = p_{t,i',j'}\}$*. If* $\bigcup_j \mathsf{S}_{i,j,t} \neq \emptyset$*, we define* $j_{i,t} \coloneqq \min(j \in [m_{C,i}]$ *s.t.* $\mathsf{S}_{i,j,t} \neq \emptyset)$*,* i' *is defined as the unique* $i' \in \mathsf{S}_{i,j_{i,t},t}$ *and* $\hat{\mathsf{v}}_{i,t} \coloneqq \mathsf{v}_{t,i'}$*. If* $\bigcup_j \mathsf{S}_{i,j,t} = \emptyset$*, we define* $\hat{\mathsf{v}}_{t,i} \coloneqq 0$*. Finally, the values* $\hat{\mathsf{v}}_{i,1}, \ldots, \hat{\mathsf{v}}_{i,T}$ *are permuted by a random permutation* π_i *for each row* $i \in [m_C]$*.*

This definition ensures that value $\hat{\mathsf{v}}_{i,t}$ is associated with delegator t such that each row in the join corresponds to T values, one for each delegator. There might be multiple possible matching rows for each delegator with one of the identifiers in c_i. In that case, we include the row that matches with $\mathsf{c}_{i,j}$ with the smallest j in the join. Since each identifier is unique in each set, there is only one identifier that matches with $\mathsf{c}_{i,j}$.

We adjust Def. 3 in case values cannot be assigned to specific delegators anymore. Therefore a row might contain multiple values of the same delegator while other delegators might not be represented with a value. Changing the definition of the join allows us to reduce the overall leakage for the $\mathsf{D}_s\mathsf{PMC}$ protocol.

Definition 4 (Multi-Key Left Join With Associated Data and Minimal Leakage Between $T+1$ Key-Value Stores). *Let for all* $t \in [T]$*,* $\mathsf{KV}_t \coloneqq \{\mathsf{p}_{t,i}, \mathsf{v}_{t,i}\}_{i\in[m_t]}$ *be a multi-key key-value store of party* P_t *that contains both key sets, i.e.,* $\mathsf{p}_{t,i} \coloneqq \{\mathsf{p}_{t,i,j}\}_{j\in[m_{t,i}]}$*, and values, i.e.,* $\mathsf{v}_{t,i}$*. Also, let* $\mathsf{KV}_C \coloneqq \{\mathsf{c}_i\}_{i\in[m_C]}$ *be a multi-key set of party* C *that contains only ordered key sets, i.e.,* $\mathsf{c}_i \coloneqq \{\mathsf{c}_{i,j}\}_{j\in[m_{C,i}]}$*. The left join between* KV_C *and* $\{\mathsf{KV}_t\}_{t\in[T]}$ *is defined as:*

$$\mathsf{KV}_C \bowtie \{\mathsf{KV}_1, \ldots, \mathsf{KV}_T\} \coloneqq (\pi_i(\hat{\mathsf{v}}_{i,1}, \ldots, \hat{\mathsf{v}}_{i,T}))_{i\in[m_C]},$$

where for each $i \in [m_C]$*,* $\hat{\mathsf{v}}_{i,t}$ *is defined as follows. Let for each* $j \in [m_{C,i}]$*,* $\mathsf{S}_{i,j} \coloneqq \{(t', i') \in ([T], [m_{t'}]) \mid \exists j' \in [m_{t',i'}]$ *s.t.* $c_{i,j} = p_{t',i',j'}\}$*. Further, we define the set of indices that have not been included in the join yet as* $\mathsf{S}_{i,j,<t} \coloneqq \mathsf{S}_{i,j} \setminus \{(t', i') \in ([T], [m_{t'}]) \mid \exists t'' < t$ *s.t.* $\hat{\mathsf{v}}_{i,t''} = \mathsf{v}_{t',i'}\}$*. If* $\bigcup_j \mathsf{S}_{i,j,<t} \neq \emptyset$*, we define* $j_{i,t} \coloneqq \min(j \in [m_{C,i}]$ *s.t.* $\mathsf{S}_{i,j,<t} \neq \emptyset)$*,* (t', i') *is defined as a random* $(t', i') \xleftarrow{R} \mathsf{S}_{i,j_{i,t},<t}$ *and* $\hat{\mathsf{v}}_{i,t} \coloneqq \mathsf{v}_{t',i'}$*. If* $\bigcup_j \mathsf{S}_{i,j,<t} = \emptyset$*, we define* $\hat{\mathsf{v}}_{t,i} \coloneqq 0$*. Finally, the values* $\hat{\mathsf{v}}_{i,1}, \ldots, \hat{\mathsf{v}}_{i,T}$ *are permuted by a random permutation* π_i *for each row* $i \in [m_C]$*.*

Def. 4 defines the join as follows. It defines value $\hat{\mathsf{v}}_{i,t}$ by matching $c_{i,j}$ for the smallest j with a match that has not yet been included in the join and takes the value of a random row i' of a random delegator t' that matches with $c_{i,j}$. If there is no such a match left, $\hat{\mathsf{v}}_{i,t}$ is defined as 0.

We provide Algs. 1 and 2 for Defs. 3 and 4 in Appendix A.

Definition 5 (Key Encapsulation Mechanism (KEM)). *A key encapsulation with security parameter κ is a triplet of algorithms* $(\mathsf{KEM.KG}, \mathsf{KEM.Enc}, \mathsf{KEM.Dec})$ *with the following syntax.*

- $\mathsf{KEM.KG}(1^\kappa)$*: On input* 1^κ *output a key pair* $(\mathsf{KEM.pk}, \mathsf{KEM.sk})$.
- $\mathsf{KEM.Enc}(\mathsf{KEM.pk})$*: On input* $\mathsf{KEM.pk}$, $\mathsf{KEM.Enc}$ *outputs an encapsulation* $\mathsf{KEM.cp}$ *and key* $\mathsf{KEM.k}$.
- $\mathsf{KEM.Dec}(\mathsf{KEM.sk}, \mathsf{KEM.cp})$*: On input* $(\mathsf{KEM.sk}, \mathsf{KEM.cp})$, $\mathsf{KEM.Dec}$ *outputs a key* $\mathsf{KEM.k}$.

For correctness, we ask that

$$\Pr[\mathsf{KEM.Dec}(\mathsf{KEM.sk}, \mathsf{KEM.cp}) = \mathsf{KEM.k}] \geq 1 - \mathsf{negl},$$

where the probability is taken over $(\mathsf{KEM.pk}, \mathsf{KEM.sk}) \leftarrow \mathsf{KEM.KG}(1^\kappa)$ *and* $(\mathsf{KEM.cp}, \mathsf{KEM.k}) \leftarrow \mathsf{KEM.Enc}(\mathsf{KEM.pk})$.

We need simulatable KEMs, which is true for commonly used KEMs. A KEM is simulatable if there exists a ppt algorithm $\mathsf{KEM.Sim}$ with $\mathsf{KEM.cp} \leftarrow \mathsf{KEM.Sim}(\mathsf{KEM.sk}, \mathsf{KEM.k})$, where $\mathsf{KEM.cp}$ is computationally indistinguishable from $\mathsf{KEM.cp}' \leftarrow \mathsf{KEM.Enc}(\mathsf{KEM.pk})$ under the constraint that $\mathsf{KEM.k} = \mathsf{KEM.Dec}(\mathsf{KEM.sk}, \mathsf{KEM.cp}')$. Further, we need standard key indistinguishability.

Definition 6 (Key Indistinguishability). *We call a KEM key indistinguishable if for any* ppt *algorithm* $\mathcal{A}$*,*

$$\left|\Pr[\mathcal{A}(\mathsf{KEM.pk}, \mathsf{KEM.cp}, \mathsf{KEM.k}) = 1] - \Pr[\mathcal{A}(\mathsf{KEM.pk}, \mathsf{KEM.cp}, u) = 1]\right| \leq \mathsf{negl},$$

where $(\mathsf{KEM.pk}, \mathsf{KEM.sk}) \leftarrow \mathsf{KEM.KG}(1^\kappa)$, $(\mathsf{KEM.cp}, \mathsf{KEM.k}) \leftarrow \mathsf{KEM.Enc}(\mathsf{KEM.pk})$ *and* $u \leftarrow \{0,1\}^*$.

Definition 7 (Secret Sharing). *We call two values* $\mathsf{sh}_1, \mathsf{sh}_2 \in \{0,1\}^*$ *a two-out-of-two* `XOR` *secret sharing of a secret value a if* $\mathsf{sh}_1 \oplus \mathsf{sh}_2 = a$ *and for* $i \in \{0,1\}$ sh_i *is uniform and independent of a.*

Secret sharing schemes allow a dealer to distribute shares of her data to multiple parties so that each share does not reveal anything about the original data [2]. In MPC, each party creates secret shares of their data and shares them with the other parties. Then, each party computes a function of the shares and combines them to reconstruct the final output. MPC utilizes secret sharing to compute arbitrary arithmetic functions as arithmetic circuits [52,19,2,32]. In this work, we utilize binary (`XOR`) secret sharing as in Def. 7, but our protocols can also support arithmetic shares. To compute arbitrary functions as arithmetic circuits, `XOR` shares can be converted to arithmetic as in [16,33].

Definition 8 (IND-CPA Security). *We call an encryption scheme indistinguishable under chosen plaintext attacks (IND-CPA secure) if for any* ppt *algorithm* $\mathcal{A}$*,*

$$\left|\Pr[\mathcal{A}(\mathsf{pk}, \mathsf{ct}_0) = 1] - \Pr[\mathcal{A}(\mathsf{pk}, \mathsf{ct}_1) = 1]\right| \leq \mathsf{negl},$$

where $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{PKE.KG}(1^\kappa)$, $(x_0, x_1)\mathcal{A}(\mathsf{pk})$, $\forall i \in \{0,1\} : \mathsf{ct}_i \leftarrow \mathsf{PKE.Enc}(\mathsf{pk}, x_i)$. *In case of a symmetric key encryption, we replace $\mathcal{A}$'s access to* pk *with access to an encryption oracle for key* sk*. We also replace* $(\mathsf{PKE.KG}, \mathsf{PKE.Enc}, \mathsf{PKE.Dec})$ *with* $(\mathsf{SKE.KG}, \mathsf{SKE.Enc}, \mathsf{SKE.Dec})$.

We include additional definitions such as the DDH assumption, pseudorandom generator, random oracle, and symmetric and public key encryption in Appendix A.

2.3 Ideal Functionality for Delegated PMC

We present the ideal functionality $\mathcal{F}_{\mathsf{DPMC}}$ for Delegated PMC in Fig. 2. In the ideal world, $\mathcal{F}_{\mathsf{DPMC}}$ is composed of a functionality for join $\mathcal{F}_{\mathsf{JOIN}}$ and a functionality for compute $\mathcal{F}_{\mathsf{CMP}}$. $\mathcal{F}_{\mathsf{JOIN}}$ gets input from party C a multi-key set KV_C and from parties P_1 to P_T multi-key key-value sets $\mathsf{KV}_1, \ldots, \mathsf{KV}_T$ with associated values $\mathsf{v}_1, \ldots, \mathsf{v}_T$ and computes a left join $\mathcal{J}$ with associated data as described in Def. 3 (or alternatively Def. 4).

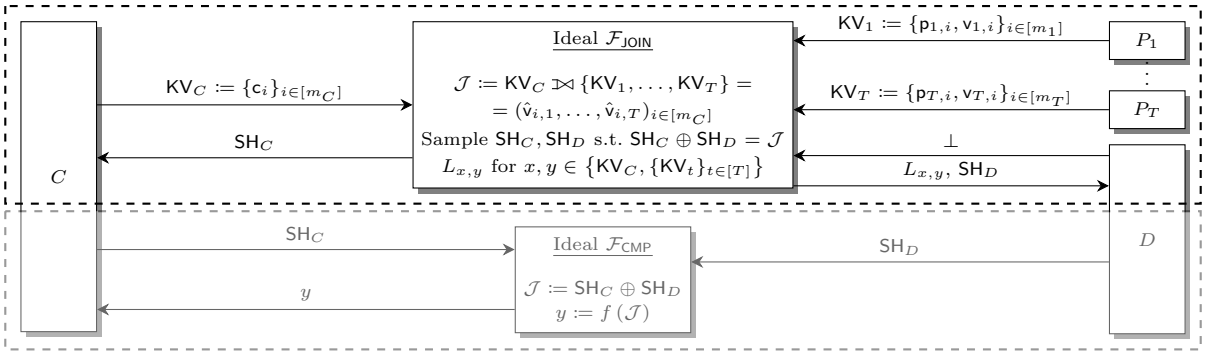


Fig. 2. Ideal functionality $\mathcal{F}_{\mathsf{DPMC}}$ of private join for compute is composed by $\mathcal{F}_{\mathsf{JOIN}}$ and $\mathcal{F}_{\mathsf{CMP}}$. Parties C and $P_1, \ldots, P_T$ provide inputs. $\mathcal{F}_{\mathsf{JOIN}}$ computes a left join with associated data $\mathcal{J} \coloneqq \mathsf{KV}_C \bowtie \{\mathsf{KV}_1, \ldots, \mathsf{KV}_T\}$ as described in Def. 3 (or alternatively Def. 4). Later on, C and D can query $\mathcal{F}_{\mathsf{CMP}}$ with their secret shares (SH_C and SH_D) and $\mathcal{F}_{\mathsf{CMP}}$ will reconstruct $\mathcal{J} \coloneqq \mathsf{SH}_C \oplus \mathsf{SH}_D$ and compute $y \coloneqq f(\mathcal{J})$ and send it to party C. Party D gets leakage $L_{x,y}$, where x and y are the sets of any party in $\{\mathsf{KV}_C, \mathsf{KV}_1, \ldots, \mathsf{KV}_T\}$.

That is, for each record c_i and for each party $t \in [T]$, $\mathcal{J}$ holds $\hat{\mathsf{v}}_{i,t}$ which represents either the associated metadata (if there was a match) or a zero (if no match was found for c_i) as $(\hat{\mathsf{v}}_{i,1}, \ldots, \hat{\mathsf{v}}_{i,T})_{i \in [m_C]}$. Next, $\mathcal{F}_{\mathsf{JOIN}}$ samples secret shares SH_C and SH_D such that $\mathcal{J} = \mathsf{SH}_C \oplus \mathsf{SH}_D$ and sends SH_C to party C and SH_D to D. Later, C and D can query $\mathcal{F}_{\mathsf{CMP}}$ with their secret shares and $\mathcal{F}_{\mathsf{CMP}}$ first reconstructs $\mathcal{J} \coloneqq \mathsf{SH}_C \oplus \mathsf{SH}_D$ and then computes $y \coloneqq f(\mathcal{J})$. Our ideal functionality $\mathcal{F}_{\mathsf{DPMC}}$ is composed of the functionality of join $\mathcal{F}_{\mathsf{JOIN}}$ and compute $\mathcal{F}_{\mathsf{CMP}}$.

Parties P_1 to P_T do not get any output from $\mathcal{F}_{\mathsf{JOIN}}$ or $\mathcal{F}_{\mathsf{CMP}}$, whereas C and D learn secret shares of the associated data of matched values from $\mathcal{F}_{\mathsf{JOIN}}$. Finally, C learns the output y from $\mathcal{F}_{\mathsf{CMP}}$ which depends on function f. Even in the ideal world, if f returns all the associated values without performing any computation (e.g., aggregation), C does not learn which value corresponds to which user, or even which of the users in KV_C have been matched. Finally, D learns a leakage function Lx, y, where x and y each represent any of KV_C or $\mathsf{KV}_1, \ldots, \mathsf{KV}_T$. We extend our $\mathcal{F}_{\mathsf{DPMC}}$ functionality to $\mathcal{F}_{\mathsf{D}_s\mathsf{PMC}}$ that limits the aforementioned leakage between sets KV_C and KV_P, where $\mathsf{KV}_P \coloneqq \{\mathsf{KV}_1, \ldots, \mathsf{KV}_T\}$. We provide a formal definition of the leakage function when introducing the different protocols. In the case of a single identifier per row, the leakage corresponds to the cardinality of the intersection.

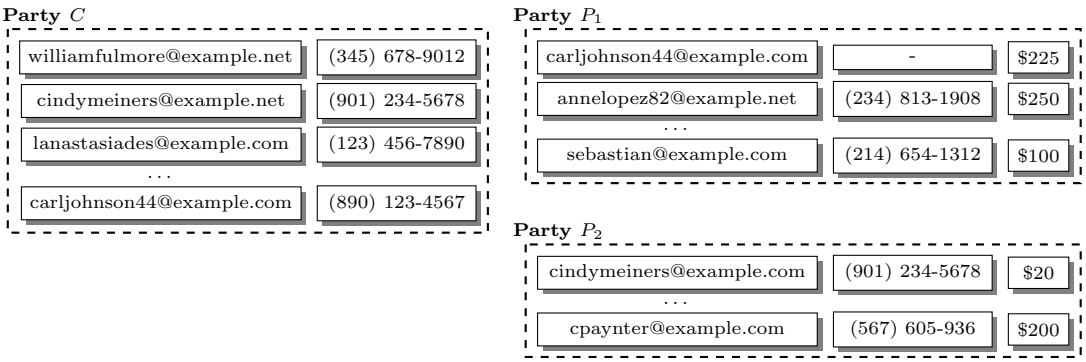
Note that the MPC functions computed in the $\mathcal{F}_{\mathsf{CMP}}$ phase should be carefully considered for privacy reasons. Without making any assumptions about the inputs, using differential privacy seems to be the only option to protect individual users being signaled out. We discuss realistic applications in Section 5.

3 Left Join Delegated PMC Protocols

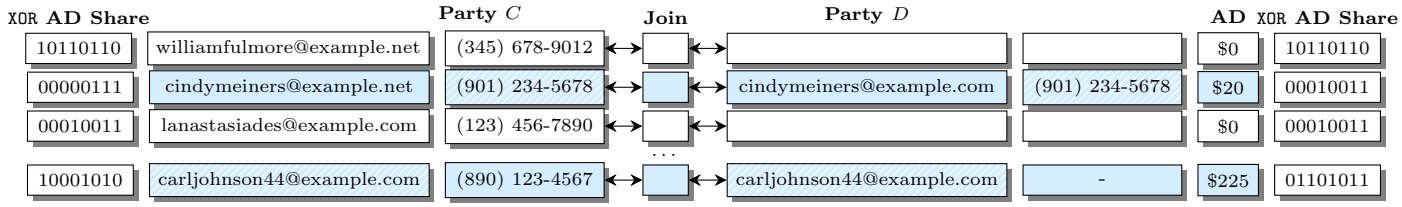
3.1 Overview

Our goal is to join records that represent the same entities across datasets that are held by different parties without revealing any information about the individual records. We focus on performing a left join between the datasets of multiple parties and computing secret shares of the associated data of the matched records so they can be fed into downstream general-purpose MPC.

We realize $\mathcal{F}_{\mathsf{DPMC}}$ with two novel protocols that compute left join with associated data. We introduce a *delegate* party D that enables multiple *delegators* (parties $P_1, \ldots, P_T$) to securely delegate their data and go offline, similarly to $\mathcal{F}_{\mathsf{JOIN}}$ in Fig. 2. Parties C and D engage in our proposed delegated protocols to privately link C's and all the delegators' records and compute secret shares of the associated data for the matched records. Both C and D learn T secret shares for each row in the left join that corresponds to either the delegators' associated data (i.e., v_t) if that row maps to a record in KV_t or a secret share of zero (if that row is only in KV_C). C also receives a mapping from its users into the join but does not learn which of its users have been matched.



(a) Input parties multi-key key-value sets containing emails, phone numbers, and dollar amounts.



(b) Multi-key left join (result of $\mathcal{F}_{\mathsf{JOIN}}$). All records of C are matched. The records of P_1 and P_2 that do not match with C do not appear in the left join.

Fig. 3. Multi-key left-join overview. Parties C and D compute a left-join of the multi-key sets of C, P_1, and P_2 and the `XOR` secret shares of the associated data (AD) of the delegators (P_1 and P_2). In (a), we show an example with three parties (C, P_1, and P_2). P_1 and P_2 have associated data (shown as \$ amounts; note that they might have more associated data). In (b), parties C and D have performed the left-join and ended up with secret shares of the associated data of the matched records (shown in blue). For readability, we show the associated data (AD) in (b) to indicate the value of the `XOR` shares, note that this remains secret.

Having the secret shares as the protocol output allows parties C and D to realize $\mathcal{F}_{\mathsf{CMP}}$ and jointly compute a function f on the secret shared associated data. An intuition of our delegated protocols is shown in Fig. 3. In Fig. 3 (a), we show the multi-key datasets of party C and two delegators P_1 and P_2. In Fig. 3 (b), we show the matching performed on both e-mail addresses and phone numbers (Def. 3), as well as the generated `XOR` shares. Interestingly, our protocols are compatible with both `XOR` and arithmetic secret shares. To keep things simple, we use `XOR` shares exclusively. Note that in Fig. 3 (b) we show the AD for readability – Party D does not learn the associated data (only the secret shares of them).

3.2 Delegated PMC (DPMC)

For simplicity, we start with a strawman DPMC protocol that does not operate over multi-key databases (e.g., has only email addresses). Our first variant for left join between $T+1$ databases is shown in Fig. 4 and consists of three stages: "key generation", "identify match", and "recover shares". Both parties C and D learn a left join size (m_C) set of `XOR` shares ($\mathcal{J}_C$ and $\mathcal{J}_D$, respectively) for each row in the join that corresponds to the delegators' associated data if that row maps one of parties' P_1 to P_T records or a secret share of zero (if that row is only present in KV_C). Additionally, C receives a mapping from its users into $\mathcal{J}_C$ but does not learn which of its users are in the intersection. The two parties can use the secret shares $\mathcal{J}_C$ and $\mathcal{J}_D$ for any general-purpose MPC computation.

Intuitively, the protocol works as follows. Each party P_t hashes its identifiers $\mathsf{p}_{t,i}$ (for each row i) using $H_{\mathbb{G}}$ and masks them with a random a_t. The associated values $\mathsf{v}_{t,i}$ are secret shared where the share for C (i.e., $\mathsf{sh}_{C,t,i}$) is the key of a KEM. Each party encrypts the shares for Party D, the mask a_t and the key encapsulation towards party D using pk_D, and sends it to C. It also sends the masked hashes of the identifiers

Setup: All parties agree on a g be a generator of a cyclic group $\mathbb{G}$ with order q where DDH is hard and hash functions $H_{\mathbb{G}}(\cdot) : \{0,1\}^* \to \mathbb{G}$,
$H(\cdot) : \{0,1\}^* \to \{0,1\}^{|\mathsf{v}_{t,i}|}$. All parties P_t have access to the public key pk_D of party D, and party D has secret key sk_D.

① Key-Generation **(Party C)**

$(\mathsf{KEM.pk}, \mathsf{KEM.sk}) \leftarrow \mathsf{KEM.KG}(1^\kappa)$

Send to P_t and D: $\mathsf{KEM.pk}$

② Identity Match **(Party P_t)**

Input: $\mathsf{KV}_t = \{(\mathsf{p}_{t,i}, \mathsf{v}_{t,i})\}_{i\in[m_t]}$ for data set size m_t.

Messages: $\mathsf{KEM.pk}$

$a_t \xleftarrow{R} \mathbb{Z}_q$, $\mathsf{sk}_t \leftarrow \mathsf{SKE.KG}(1^\kappa)$ ▷ Random scalar a_t and secret key sk_t.

$\mathsf{cta}_t \coloneqq \mathsf{PKE.Enc}(\mathsf{pk}_D, \mathsf{sk}_t)$, $\mathsf{ctb}_t \coloneqq \mathsf{SKE.Enc}(\mathsf{sk}_t, a_t)$

For $i \in [m_t]$: ▷ For each row in KV_t.

  $(\mathsf{KEM.cp}_{t,i}, \mathsf{KEM.k}_{t,i}) \leftarrow \mathsf{KEM.Enc}(\mathsf{KEM.pk})$ ▷ New encaps.

  $\mathsf{ha}_{t,i} \coloneqq H_{\mathbb{G}}(\mathsf{p}_{t,i})^{a_t}$ ▷ Hash and exponentiate to a_t.

  $\mathsf{sh}_{C,t,i} \coloneqq \mathsf{KEM.k}_{t,i}$ ▷ Share of $\mathsf{v}_{t,i}$ for party C.

  $\mathsf{sh}_{D,t,i} \coloneqq \mathsf{v}_{t,i} \oplus \mathsf{sh}_{C,t,i}$ ▷ Share of $\mathsf{v}_{t,i}$ for party D.

  $\mathsf{ctc}_{t,i} \coloneqq \mathsf{SKE.Enc}(\mathsf{sk}_t, (\mathsf{KEM.cp}_{t,i}, \mathsf{sh}_{D,t,i}))$

Send to C: $\mathsf{cta}_t, \mathsf{ctb}_t$ and $\{\{\mathsf{ha}_{t,i}\}, \mathsf{ctc}_{t,i}\}_{i\in[m_t]}$

③ Identity Match **(Party C)**

Input: $\mathsf{KV}_C = \{\mathsf{c}_i\}_{i\in[m_C]}$ for data set size m_C.

Messages: $\{\mathsf{cta}_t, \mathsf{ctb}_t, \{\{\mathsf{ha}_{t,i}\}, \mathsf{ctc}_{t,i}\}_{i\in[m_t]}\}_{t\in[T]}$

$a_C \xleftarrow{R} \mathbb{Z}_q$ ▷ Random scalar a_C.

For $t \in [T], i \in [m_t]$: ▷ For each P_t and each row.

  $\mathsf{hca}_{t,i} \coloneqq (\mathsf{ha}_{t,i})^{a_C}$ ▷ Hash and exponentiate P_t's data to a_C.

Pick random permutation π, $\hat{t} \coloneqq \pi(t)$.

For $i \in [m_C]$: ▷ For each row in KV_C.

  $\mathsf{h}_{C,i} \coloneqq H_{\mathbb{G}}(\mathsf{c}_i)^{a_C}$ ▷ Hash and exponentiate own data to a_C.

Send to D: $\{\mathsf{h}_{C,i}\}_{i\in[m_C]}$ and $\{\mathsf{cta}_{\hat{t}}, \mathsf{ctb}_{\hat{t}}, \{\mathsf{hca}_{\hat{t},i}, \mathsf{ctc}_{\hat{t},i}\}_{i\in[m_{\hat{t}}]}\}_{\hat{t}\in[T]}$

④ Identity Match and Recover Shares **(Party D)**

Messages: $\mathsf{KEM.pk}$, $\{\mathsf{h}_{C,i}\}_{i\in[m_C]}$ and $\{\mathsf{cta}_{\hat{t}}, \mathsf{ctb}_{\hat{t}}, \{\mathsf{hca}_{\hat{t},i}, \mathsf{ctc}_{\hat{t},i}\}_{i\in[m_{\hat{t}}]}\}_{\hat{t}\in[T]}$

For $\hat{t} \in [T]$: ▷ For each delegator party $P_{\hat{t}}$.

  $\mathsf{sk}_{\hat{t}} \coloneqq \mathsf{PKE.Dec}(\mathsf{sk}_D, \mathsf{cta}_{\hat{t}})$

  $a_{\hat{t}} \coloneqq \mathsf{SKE.Dec}(\mathsf{sk}_{\hat{t}}, \mathsf{ctb}_{\hat{t}})$

  For $i \in [m_{\hat{t}}]$: ▷ For each row in $\mathsf{KV}_{\hat{t}}$.

    $(\mathsf{KEM.cp}_{\hat{t},i}, \mathsf{sh}_{D,\hat{t},i}) \coloneqq \mathsf{SKE.Dec}(\mathsf{sk}_{\hat{t}}, \mathsf{ctc}_{\hat{t},i})$

    $\mathsf{hc}_{\hat{t},i} \coloneqq \mathsf{hca}_{\hat{t},i}^{1/a_{\hat{t}}}$ ▷ Remove $a_{\hat{t}}$.

Join $\mathcal{J} \coloneqq (\mathsf{h}_{C,i})_{i\in[m_C]} \bowtie (\mathsf{hc}_{t,i})_{t\in[T],i\in[m_t]}$ ▷ Details in Alg. 1.

For each row i and delegator $\hat{t}$ in $\mathcal{J}$: ▷ For each row in the join.

  If record matched:

    $\widehat{\mathsf{KEM.cp}}_{i,\hat{t}} \coloneqq \mathsf{KEM.cp}_{\hat{t},i'}$ ▷ Use encaps. from $P_{\hat{t}}$.

    $\widehat{\mathsf{sh}}_{D,i,\hat{t}} \coloneqq \mathsf{sh}_{D,\hat{t},i'}$ ▷ Use share of $\mathsf{v}_{\hat{t},i}$ generated by $P_{\hat{t}}$.

  Else: ▷ no match found

    $(\widehat{\mathsf{KEM.cp}}_{i,\hat{t}}, \mathsf{KEM.k}_{i,\hat{t}}) \leftarrow \mathsf{KEM.Enc}(\mathsf{KEM.pk})$ ▷ New encaps.

    $\widehat{\mathsf{sh}}_{D,i,\hat{t}} \coloneqq \mathsf{KEM.k}_{i,\hat{t}}$ ▷ Use share of 0.

Pick m_C random permutations $\{\pi_i\}_{i\in[m_C]}$.

$\mathcal{J}_D \coloneqq (\pi_i(\{\widehat{\mathsf{sh}}_{D,i,\hat{t}}\}_{\hat{t}\in[T]}))_{i\in[m_C]}$ ▷ D's permuted XOR shares.

Send to C: $\{\pi_i(\{\widehat{\mathsf{KEM.cp}}_{i,\hat{t}}\}_{\hat{t}\in[T]})\}_{i\in[m_C]}$

⑤ Recover Shares **(Party C)**

Input: $\mathsf{KEM.sk}$

Messages: $\{\widehat{\mathsf{KEM.cp}}_{i,\hat{t}}\}_{i\in[m_C],\hat{t}\in[T]}$

For $i \in [m_C], \hat{t} \in [T]$: ▷ For each row and each delegator.

  $\widehat{\mathsf{sh}}_{C,i,\hat{t}} \coloneqq \mathsf{KEM.Dec}(\mathsf{KEM.sk}, \widehat{\mathsf{KEM.cp}}_{i,\hat{t}})$ ▷ Get $\mathsf{KEM.k}_{i,\hat{t}}$.

$\mathcal{J}_C \coloneqq (\widehat{\mathsf{sh}}_{C,i,\hat{t}})_{i\in[m_C],\hat{t}\in[T]}$ ▷ Aligned with $(\mathsf{c}_i)_{i\in[m_C]}$

Fig. 4. Single-key DPMC. Party C and the delegators P_1 to P_T compute the left-join of their records with the help of D. Parties C and D receive $\mathcal{J}_C$ and $\mathcal{J}_D$, respectively. These sets contain XOR secret shares for each row in the join. For each delegator P_t, if a row is in the intersection, the parties hold XOR shares of the delegator's associated data, otherwise, XOR shares zero. Party C additionally learns a mapping from its users into the join a but does not learn which of its users have been matched.

(i.e., $H_{\mathbb{G}}(\mathsf{p}_{t,i})^{a_t}$) to C. Note that this does not leak any information to C since $H_{\mathbb{G}}(\mathsf{p}_{t,i})^{a_t}$ could be seen as a PRF evaluation and is therefore pseudorandom based on DDH.

Party C permutes the messages and uses a random $a_C \xleftarrow{R} \mathbb{Z}_q$ to compute $\mathsf{hca}_{t,i} \coloneqq H_{\mathbb{G}}(\mathsf{p}_{t,i})^{a_t \cdot a_C}$. a_C can be seen as a PRF key. It forwards the permuted messages including $\mathsf{hca}_{t,i}$ and sends the PRF evaluation of its own identifiers, i.e., $\mathsf{hc}_i \coloneqq H_{\mathbb{G}}(\mathsf{c}_i)^{a_C}$ to D.

Party D decrypts all the ciphertexts and unmasks $H_{\mathbb{G}}(\mathsf{p}_{t,i})^{a_t \cdot a_C}$ to $H_{\mathbb{G}}(\mathsf{p}_{t,i})^{a_C}$ using a_t. It then matches the results with the hc_i's sent by Party C. If there is a match, it just forwards the key encapsulation $\mathsf{KEM.cp}_{\hat{t},i'}$ from delegator P_t and uses the decrypted share $\mathsf{sh}_{D,t,i}$ as its own share. Otherwise, it generates a new encapsulation and uses the generated key $\mathsf{KEM.k}_{i,\hat{t}}$ as its own share. In this step, we do not leak C's share and therefore value $\mathsf{v}_{t,i}$ to Party D, due to the key indistinguishability of the key encapsulation.

In the final step, C uses the secret key of the key encapsulation received by D to recover its own shares. Observe that for the unmatched records, we get secret shares of zero as $\mathcal{J}_C \oplus \mathcal{J}_D = \widehat{\mathsf{sh}}_{C,i,\hat{t}} \oplus \widehat{\mathsf{sh}}_{D,i,\hat{t}} = \mathsf{KEM.k}_{i,\hat{t}} \oplus \mathsf{KEM.k}_{i,\hat{t}} = 0$, while for the matched records we get secret shares of the delegators associated data as $\mathcal{J}_C \oplus \mathcal{J}_D = \widehat{\mathsf{sh}}_{C,i,\hat{t}} \oplus \widehat{\mathsf{sh}}_{D,i,\hat{t}} = \mathsf{KEM.k}_{i,\hat{t}} \oplus \mathsf{sh}_{D,\hat{t},i} = \mathsf{KEM.k}_{i,\hat{t}} \oplus \mathsf{v}_{t,i} \oplus \mathsf{sh}_{C,\hat{t},i} = \mathsf{v}_{t,i}$. Party D cannot distinguish shares of $\mathsf{v}_{t,i}$ from shares of 0 since the encapsulations generated by parties P_1 to P_T have the same distribution as the ones generated by D.

Leakage. We define DPMC's leakage in Def. 9, where D learns the sizes of the intersection between each two parties. For instance, for parties C, P_1, and P_2, party D will learn $|\mathsf{KV}_C \cap \mathsf{KV}_1|$, $|\mathsf{KV}_C \cap \mathsf{KV}_2|$, and $|\mathsf{KV}_1 \cap \mathsf{KV}_2|$ but without knowing which party is P_1 and which is P_2 due to the permutation performed by C. With multiple keys, D will also learn a graph of matches as defined by $L_{x,y}$ next. We give a formal security theorem (Theorem 1) and prove it in Appendix D.1.

Definition 9 (DPMC Leakage). *Given KV_C and $\mathsf{KV}_1, \ldots, \mathsf{KV}_T$, the leakage $L_{x,y}$ of the ideal functionality in Fig. 2 for the DPMC protocol in Fig. 4 is defined as follows. Define $\mathsf{KV}_{u,C}$ by replacing $\mathsf{c}_{i,j} \in \mathsf{KV}_C$ with $u_{i,j} \xleftarrow{R} \{0,1\}^\kappa$. Define $\mathsf{KV}_{u,t}$ by replacing $\mathsf{p}_{t,i,j} \in \mathsf{KV}_t$ with $u_{i',j'}$ if there exist t', i', j' with $\mathsf{p}_{t,i,j} = \mathsf{c}_{i',j'}$ or an already replaced $\mathsf{p}_{t',i',j'}$ with $\mathsf{p}_{t,i,j} = \mathsf{p}_{t',i',j'}$, otherwise replace it with $u'_{t,i,j} \xleftarrow{R} \{0,1\}^\kappa$. $L_{x,y} := \{(C, \mathsf{KV}_{u,C}), \pi(t, \mathsf{KV}_{u,t})_{t\in[T]}\}$.*

Theorem 1. *Let the secret key encryption and the PKE scheme be IND-CPA secure, the KEM simulatable and key indistinguishable, and the DDH assumption hold.*

Then, the protocol in Fig. 4 securely realizes ideal functionality in Fig. 2 for the join defined in Def. 3 for semi-honest corruption of one of the two parties C, D and any amount of parties P_1 to P_T. In case of a corruption of D, the leakage graph of Def. 9 is leaked.

3.3 Rerandomizable Encrypted OPRF (EO)

OPRFs allow a client to obliviously evaluate a function PRF on their private input x with the server's secret key sk (i.e., $\mathsf{PRF}_{sk}(x)$) [11]. We introduce a new rerandomizable encrypted OPRF (EO) primitive with more powerful functionality that allows: (a) multiple input providers to encrypt their inputs, (b) an output receiver to shuffle and rerandomize the ciphertexts, (c) a server to obliviously evaluate a PRF on encrypted as well as plaintext identifiers, and (d) the output receiver to decrypt the encrypted PRF evaluations. Our EO primitive consists of a collection of seven algorithms:

Definition 10 (EO). *A rerandomizable encrypted OPRF (EO) parameterized with security parameter κ is a collection of algorithms* $(\mathsf{KG}, \mathsf{EKG}, \mathsf{Eval}, \mathsf{Enc}, \mathsf{Rnd}, \mathsf{OEval}, \mathsf{Dec})$ *with the following syntax.*

- $\mathsf{KG}(1^\kappa)$*: On input 1^κ output a public key, secret key pair* $(\mathsf{pk}, \mathsf{sk})$.
- $\mathsf{EKG}(1^\kappa)$*: On input 1^κ output a public function key, evaluation key pair* $(\mathsf{pf}, \mathsf{ek})$.
- $\mathsf{Eval}(\mathsf{ek}, x)$*: On input* (ek, x)*, output a* PRF *output y.*
- $\mathsf{Enc}(\mathsf{pk}, \mathsf{pf}, x)$*: On input* $(\mathsf{pk}, \mathsf{pf}, x)$*, output a ciphertext* ct.
- $\mathsf{Rnd}(\mathsf{pk}, \mathsf{pf}, \mathsf{ct})$*: On input* $(\mathsf{pk}, \mathsf{pf}, \mathsf{ct})$*, output a ciphertext* ct'.
- $\mathsf{OEval}(\mathsf{ek}, \mathsf{ct})$*: On input* $(\mathsf{ek}, \mathsf{ct})$*, output evaluated ciphertext* ect.
- $\mathsf{Dec}(\mathsf{sk}, \mathsf{ect})$*: On input* $(\mathsf{sk}, \mathsf{ect})$*, output y.*

For correctness, we ask that for any $x \in \{0,1\}^*$,

$$\Pr[\mathsf{Dec}(\mathsf{sk}, \mathsf{OEval}(\mathsf{ek}, \mathsf{Rnd}(\mathsf{pk}, \mathsf{pf}, \mathsf{Enc}(\mathsf{pk}, \mathsf{pf}, x))) = \mathsf{Eval}(\mathsf{ek}, x)] \geq 1 - \mathsf{negl},$$

where $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KG}(1^\kappa)$ *and* $(\mathsf{pf}, \mathsf{ek}) \leftarrow \mathsf{EKG}(1^\kappa)$.

We use EO as shown in Fig. 5. Each *Input Provider* t invokes EO.Enc to encrypt identifier x_t. Afterward, an *Output Receiver* rerandomizes and shuffles the ciphertexts using EO.Rnd. We remark that for security, we require that neither possession of EO.sk nor EO.ek is sufficient to distinguish encryptions of two different messages. After the shuffle, the *Server* uses EO.OEval to homomorphically evaluate a PRF on the encrypted identifier. The Server also evaluates the PRF on plaintext identifiers without knowledge of EO.sk by using EO.Eval and knowledge of EO.ek. Finally, the Output Receiver uses EO.Dec to decrypt the PRF evaluation. Observe that both the Server and the Output Receiver end up with the same PRF evaluation y for the same input x (or x_t). In order to have a PRF, we require that the EO.Eval outputs are pseudorandom given EO.pk, EO.sk, and EO.pf. In Appendix B, we define several security notions and show how to construct this primitive from DDH.

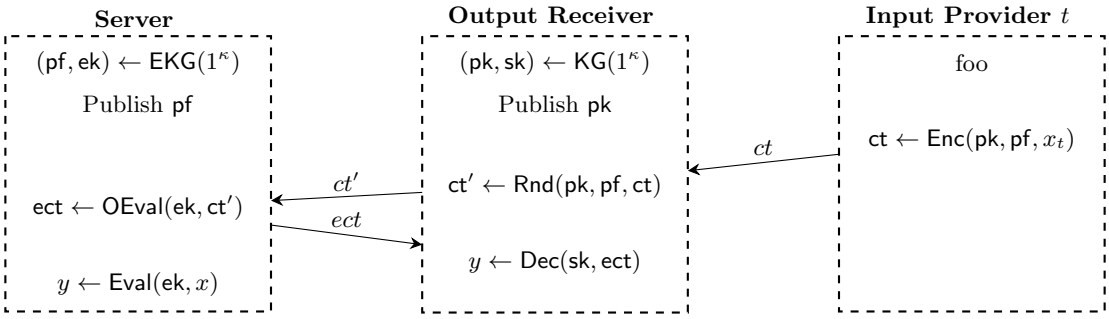


Fig. 5. Rerandomizable Encrypted OPRF (EO).

3.4 DPMC with Secure Shuffling ($\mathsf{D}_s\mathsf{PMC}$)

In DPMC, party D performs the left join on the hashed and exponentiated data between C and multiple delegators denoted as P_t. This process enables D to learn the full bipartite graph of correlations of matches up to an isomorphism due to shared identifiers. We address this issue with an enhanced version called $\mathsf{D}_s\mathsf{PMC}$ that utilizes our novel EO scheme and employs two delegates: party D and a new *shuffler* party S. $\mathsf{D}_s\mathsf{PMC}$ relies on EO to perform a secure three-party shuffling protocol between C, D, and S that combines and shuffles the data from all delegator parties P_t. In the process of secure shuffling, the data from the delegators are reordered in a way that no single party knows the applied permutation. Additionally, the delegators' data undergo two forms of rerandomization. First, the encrypted identifiers are refreshed with new ciphertexts using the EO.Rnd algorithm, generating new ciphertexts that correspond to the same plaintexts. Second, the secret shares of the associated data are reshared, creating new secret shares of the same plaintext values. Notably, these rerandomization steps do not reveal the underlying data and are meant to break any link between the data that the delegators provide and the data that are used for the join and the secret sharing. This way, the leakage to party D is only between C's data and the combined data of parties P_1 to P_T, contrary to the pairwise leakages of DPMC. Since C combines the inputs of all delegators, party D (who performs the join) only sees two encrypted datasets (i.e., encrypted KV_C and encrypted KV_P).

Due to our shuffling and rerandomization steps, in a potential corruption of the delegators and one of C, D, S, the corrupted parties cannot infer any information as the data have been permuted and rerandomized. Our shuffling scheme is secure in the honest-majority setting, which is the case with multiple applications from both academia [1,15,40] and industry. For instance, Mozilla recently deployed a service that relies on the Prio protocol to collect telemetry data about Firefox [30], while Crypten [33] and TF Encrypted [17] build privacy-preserving machine learning frameworks for PyTorch and TensorFlow, respectively. We delve into the details of the security of $\mathsf{D}_s\mathsf{PMC}$ in Appendix D.2.

$\mathsf{D}_s\mathsf{PMC}$ follows a similar approach as DPMC, with the difference that it leverages our EO primitive. We formally present our $\mathsf{D}_s\mathsf{PMC}$ protocol in Fig. 6; intuitively, it works as follows. The delegator parties use EO to encrypt their identifiers and generate XOR shares as in the DPMC protocol. Then, the delegators encrypt the shares for party D using pk_D and send them to Party C along with C's shares and all of the ciphertexts. C then forwards D's encrypted shares to Party D who decrypts them. Then parties C, D, and S run a secure shuffling protocol in which C receives rerandomized EO ciphertexts ($\widetilde{\mathsf{EO.ct}}$), S obtains the rerandomized shares for C ($\widetilde{\mathsf{sh}}_C$), and D receives its randomized shares ($\widetilde{\mathsf{sh}}_D$).

Next, S generates new key encapsulations and uses C's shares to generate new shares for D. It sends the updated $\overline{\mathsf{sh}}_{D,i}$ to party D that allows D to adjust their shares to be consistent with the new shares of C. S also sends the encapsulations (KEM.cp) to D.

Party C proceeds by homomorphically evaluating the PRF on the EO ciphertexts and the PRF on its own identifiers $\mathsf{c}_{i,j}$ and sends the outcomes to D. Recall from Fig. 5 that all the output receiver (D in this case) needs to do now is to decrypt the evaluated EO ciphertexts and compute the matches. D computes the left join with associated data, where for each matched record it ends up with T shares of either the delegators' associated data or zero. Note that D does not know which decrypted PRF identifier belongs to which delegator. The matching logic is described in more detail in Def. 4 in the Appendix A. Similar to DPMC, D it replaces the encapsulation with a fresh encapsulation when no match is found and keeps KEM.k as its

Setup: All parties P_t have access to the public key pk_D of party D, party D has secret key sk_D. $M \coloneqq \sum_{t=1}^{T} m_t$.

① Key-Generation **(Party C)**

$(\mathsf{KEM.pk}, \mathsf{KEM.sk}) \leftarrow \mathsf{KEM.KG}(1^\kappa)$
$(\mathsf{EO.pf}, \mathsf{EO.ek}) \leftarrow \mathsf{EO.EKG}(1^\kappa)$
Send to P_t, S, D: $\mathsf{KEM.pk}, \mathsf{EO.pf}$

② Key-Generation **(Party D)**

$(\mathsf{EO.pk}, \mathsf{EO.sk}) \leftarrow \mathsf{EO.KG}(1^\kappa)$
Send to P_t: $\mathsf{EO.pk}$

③ Identity Match **(Party P_t)**

Input: $\mathsf{KV}_t = \{\{\mathsf{p}_{t,i,j}\}_{j\in[m_{t,i}]}\mathsf{v}_{t,i}\}_{i\in[m_t]}$ for data set size m_t.
Messages: $\mathsf{EO.pk}, \mathsf{EO.pf}$

$\mathsf{seed}_t \xleftarrow{R} \{0,1\}^\kappa$ ▷ Random seed_t.
$\mathsf{cta}_t \coloneqq \mathsf{PKE.Enc}(\mathsf{pk}_D, \mathsf{seed}_t)$
$(\mathsf{sh}_{D,t,1}, \ldots \mathsf{sh}_{D,t,m_t}) \xleftarrow{R} \mathsf{PRG}(\mathsf{seed}_t)$ ▷ Share of $\mathsf{v}_{t,i}$ for party D.
For $i \in [m_t]$: ▷ For each row in KV_t.
  For $j \in [m_{t,i}]$: ▷ For each column.
    $\mathsf{EO.ct}_{t,i,j} \leftarrow \mathsf{EO.Enc}(\mathsf{EO.pk}, \mathsf{EO.pf}, \mathsf{p}_{t,i,j})$ ▷ Encrypt data using EO.
  $\mathsf{sh}_{C,t,i} \coloneqq \mathsf{v}_{t,i} \oplus \mathsf{sh}_{D,t,i}$ ▷ Share of $\mathsf{v}_{t,i}$ for party C.
Send to C: $\mathsf{cta}_t, \{\{\mathsf{EO.ct}_{t,i,j}\}_{j\in[m_{t,i}]}, \mathsf{sh}_{C,t,i}\}_{i\in[m_t]}$

④ Forward Shares to D **(Party C)**

Messages: $\{\{\{\mathsf{EO.ct}_{t,i,j}\}_{j\in[m_{t,i}]}, \mathsf{sh}_{C,t,i}\}_{i\in[m_t]}, \mathsf{cta}_t, \}_{t\in[T]}$
Send to D: $\{\mathsf{cta}_t\}_{t\in[T]}$

⑤ Reconstruct Shares **(Party D)**

Messages: $\{\mathsf{cta}_t\}$ for all T parties P

For $t \in [T]$: ▷ For each delegator P_t
  $\mathsf{seed}_t \coloneqq \mathsf{PKE.Dec}(\mathsf{sk}_D, \mathsf{cta}_t)$ ▷ Get seed seed_t.
  $\mathsf{sh}_{D,t,1}, \ldots, \mathsf{sh}_{D,t,m_t} \coloneqq \mathsf{PRG}(\mathsf{seed}_t)$ ▷ Share of $\mathsf{v}_{t,i}$ for party D.

⑥ Shuffling – Appendix C **(Parties C, S, D)**

Note: The $\mathsf{EO.ct}$ ciphertexts as well as the sh_C and sh_D secret shares are: a) reordered such that no party knows the permutation, and b) rerandomized to $\widetilde{\mathsf{EO.ct}}$, $\widetilde{\mathsf{sh}_C}$, and $\widetilde{\mathsf{sh}_D}$. These rerandomizations correspond to fresh encryptions and fresh secret shares of the same underlying data.

C **Input:** $\{\{\mathsf{EO.ct}_{t,i,j}\}_{j\in[m_{t,i}]}, \mathsf{sh}_{C,t,i}\}_{i\in[m_t],t\in[T]}$
S **Input:** $-$
D **Input:** $\{\mathsf{sh}_{D,t,i}\}_{i\in[m_t],t\in[T]}$
C **receives output:** $\{\widetilde{\mathsf{EO.ct}}_{i,j}\}_{\hat{i}\in[M],j\in[m_{\hat{i}}]}$ ▷ Rerandomized ciphertexts.
S **receives output:** $\{\widetilde{\mathsf{sh}}_{C,\hat{i}}\}_{\hat{i}\in[M]}$ ▷ Rerandomized shares for party C.
D **receives output:** $\{\widetilde{\mathsf{sh}}_{D,\hat{i}}\}_{\hat{i}\in[M]}$ ▷ Rerandomized shares for party D.

⑦ Mask Shares **(Party S)**

Input: $\{\widetilde{\mathsf{sh}}_{C,\hat{i}}\}_{\hat{i}\in[M]}$
Messages: $\mathsf{KEM.pk}$

For i in $[M]$: ▷ For all delegators rows.
  $(\mathsf{KEM.cp}_i, \mathsf{KEM.k}_i) \leftarrow \mathsf{KEM.Enc}(\mathsf{KEM.pk})$ ▷ New encaps.
  $\overline{\mathsf{sh}}_{D,i} \coloneqq \widetilde{\mathsf{sh}}_{C,i} \oplus \mathsf{KEM.k}_i$ ▷ Updated shares for party D.
Send to D: $\{\mathsf{KEM.cp}_{\hat{i}}, \overline{\mathsf{sh}}_{D,\hat{i}}\}_{\hat{i}\in[M]}$

⑧ Prepare Match Keys **(Party C)**

Input: $\mathsf{KV}_C = \{(\mathsf{c}_{i,j})_{j\in[m_{C,i}]}\}_{i\in[m_C]}$, $\mathsf{EO.ek}$ and $\{\widetilde{\mathsf{EO.ct}}_{i,j}\}_{\hat{i}\in[M],j\in[m_{\hat{i}}]}$.

For $i \in [M], j \in [m_i]$: ▷ For all delegators rows and columns.
  $\mathsf{EO.ect}_{i,j} \coloneqq \mathsf{EO.OEval}(\mathsf{EO.ek}, \widetilde{\mathsf{EO.ct}}_{i,j})$
For $i \in [m_C]$, $j \in [m_{C,i}]$: ▷ For all rows and columns in KV_C.
  $\mathsf{h}_{C,i,j} \coloneqq \mathsf{EO.Eval}(\mathsf{EO.ek}, \mathsf{c}_{i,j})$
  Use $\mathsf{c}_i \coloneqq (\mathsf{c}_{i,j})_{j\in[m_{C,i}]}$ to order $(\mathsf{h}_{C,i,j})_{j\in[m_{C,i}]}$.
Send to D: $(\mathsf{h}_{C,i,j})_{i\in[m_C],j\in[m_{C,i}]}$, $\{\mathsf{EO.ect}_{\hat{i},j}\}_{\hat{i}\in[M],j\in[m_{\hat{i}}]}$

⑨ Identity Match and Recover Shares **(Party D)**

Input: $\mathsf{EO.sk}$ and $\{\widetilde{\mathsf{sh}}_{D,\hat{i}}\}_{\hat{i}\in[M]}$
Messages: $\mathsf{KEM.pk}$, $\{\mathsf{KEM.cp}_{\hat{i}}, \overline{\mathsf{sh}}_{D,\hat{i}}\}_{\hat{i}\in[M]}$, $\{\mathsf{h}_{C,i,j}\}_{i\in[m_C],j\in[m_{C,i}]}$, $\{\mathsf{EO.ect}_{\hat{i},j}\}_{\hat{i}\in[M],j\in[m_{\hat{i}}]}$

For $i \in [M], j \in [m_i]$: ▷ For all delegators rows and columns.
  $\mathsf{h}_{i,j} \coloneqq \mathsf{EO.Dec}(\mathsf{EO.sk}, \mathsf{EO.ect}_{i,j})$
$\mathcal{J} \coloneqq (\mathsf{h}_{C,i,j})_{i\in[m_C],j\in[m_{C,i}]} \rtimes\!\bowtie (\mathsf{hc}_{i,j})_{i\in[M],j\in[M_i]}$ ▷ Details in Alg. 2.
For each row i in $\mathcal{J}$, repeat for $t \in [T]$:▷ For each row in the join.
  If record matched:
    $\widehat{\mathsf{KEM.cp}}_i \coloneqq \mathsf{KEM.cp}_{i'}$ ▷ Use encaps. from $P_{\hat{i}}$.
    $\widehat{\mathsf{sh}}_{D,i,t} \coloneqq \widetilde{\mathsf{sh}}_{D,i'} \oplus \overline{\mathsf{sh}}_{D,i'}$ ▷ Final shares for party D.
  Else: ▷ no match found
    $(\widehat{\mathsf{KEM.cp}}_i, \mathsf{KEM.k}_i) \leftarrow \mathsf{KEM.Enc}(\mathsf{KEM.pk})$ ▷ New encaps.
    $\widehat{\mathsf{sh}}_{D,i,t} \coloneqq \mathsf{KEM.k}_i$ ▷ Use share of 0.
Pick m_C random permutations $\{\pi_i\}_{i\in[m_C]}$.
$\mathcal{J}_D \coloneqq (\pi_i(\{\widehat{\mathsf{sh}}_{D,i,t}\}_{t\in[T]}))_{i\in[m_C]}$ ▷ D's permuted $\mathtt{XOR}$ shares.
Send to C: $\{\pi_i(\{\widehat{\mathsf{KEM.cp}}_{i,t}\}_{t\in[T]})\}_{i\in[m_C]}$

⑩ Recover Shares **(Party C)**

Input: $\mathsf{KEM.sk}$
Messages: $\{\widehat{\mathsf{KEM.cp}}_{i,t}\}_{i\in[m_C],t\in[T]}$

For $i \in [m_C], t \in [T]$
  $\widehat{\mathsf{sh}}_{C,i,t} \coloneqq \mathsf{KEM.Dec}(\mathsf{KEM.sk}, \widehat{\mathsf{KEM.cp}}_{i,t})$ ▷ $(\mathsf{KEM.k}_{i,t})$ Shares for C.
$\mathcal{J}_C \coloneqq (\widehat{\mathsf{sh}}_{C,i,t})_{i\in[m_C],t\in[T]}$ ▷ Aligned with $(\mathsf{c}_i)_{i\in[m_C]}$

Fig. 6. Multi-key $\mathsf{D_sPMC}$. This protocol uses two delegate parties (S and D) and is based on secure shuffling and EO.

share $\widehat{\mathsf{sh}}_D$. It then forwards the encapsulations $\widehat{\mathsf{KEM.cp}}$ to C. Party C finalizes the protocol by recovering the encapsulated keys and using them as its shares $\widehat{\mathsf{sh}}_C$. Observe that for the unmatched records, C and D end up with secret shares of zero as $\mathcal{J}_C \oplus \mathcal{J}_D = \widehat{\mathsf{sh}}_{C,i,\hat{t}} \oplus \widehat{\mathsf{sh}}_{D,i,\hat{t}} = \mathsf{KEM.k}_{i,\hat{t}} \oplus \mathsf{KEM.k}_{i,\hat{t}} = 0$, while for the matched records we get secret shares of the delegators' associated data as $\mathcal{J}_C \oplus \mathcal{J}_D = \widehat{\mathsf{sh}}_{C,i,\hat{t}} \oplus \widehat{\mathsf{sh}}_{D,i,\hat{t}} = \mathsf{KEM.k}_i \oplus \widetilde{\mathsf{sh}}_{D,i} \oplus \overline{\mathsf{sh}}_{D,i} = \mathsf{KEM.k}_i \oplus (\mathsf{sh}_{D,i} \oplus \widetilde{\mathsf{sh}}_{C,i}) \oplus \mathsf{KEM.k}_i = \mathsf{sh}_{D,i} \oplus \mathsf{sh}_{C,i}) = \mathsf{v}_i$.

We describe our protocol's leakage in Def. 11. $\mathsf{D}_s\mathsf{PMC}$ limits the leakage of DPMC from pairwise intersection sizes between each party to one intersection size between party C and the union of all delegators. For instance, for parties C, P_1, and P_2, party D will learn $|\mathsf{KV}_C \cap \mathsf{KV}_P|$, where $\mathsf{KV}_P \coloneqq \{\mathsf{KV}_1 \cup \mathsf{KV}_2\}$. Notably, these intersection sizes also contain the number of times that keys are matched (i.e., 1 to T). In case multiple keys are used, D will additionally learn a graph of matches as defined by $L_{x,y}$ in Def. 11. We provide the security of $\mathsf{D}_s\mathsf{PMC}$ in Theorem 2 and prove it in Appendix D.2. Note that we do not need ciphertext indistinguishability for the secret key owner (Lemma 4) since D does not handle any EO ciphertexts, only evaluated ciphertext. This might change when a different shuffle protocol is used.

Definition 11 (D_sPMC Leakage). *Given KV_C and $\mathsf{KV}_1, \ldots, \mathsf{KV}_T$, the leakage $L_{x,y}$ of the ideal functionality in Fig. 2 for the D_sPMC protocol in Fig. 6 is defined as follows. Merge $\mathsf{KV}_1, \ldots, \mathsf{KV}_T$ to $\mathsf{KV}_P \coloneqq \bigcup_{t\in[T]} \mathsf{KV}_t$. Define $\mathsf{KV}_{u,C}$ by replacing $\mathsf{c}_{i,j} \in \mathsf{KV}_C$ with $u_{i,j} \xleftarrow{R} \{0,1\}^\kappa$. Define $\mathsf{KV}_{u,P}$ by replacing $\mathsf{p}_{i,j} \in \mathsf{KV}_P$ with $u_{i',j'}$ if there exists an i', j' pair with $\mathsf{p}_{i,j} = \mathsf{c}_{i',j'}$ or an already replaced $\mathsf{p}_{i',j'}$ with $\mathsf{p}_{i,j} = \mathsf{p}_{i',j'}$, otherwise replace it with $u'_{i,j} \xleftarrow{R} \{0,1\}^\kappa$. $L_{x,y} \coloneqq \{(C, \mathsf{KV}_{u,C}), (D, \mathsf{KV}_{u,P})\}$.*

Theorem 2. *Let PKE be an IND-CPA secure and correct PKE scheme, KEM a correct and key-indistinguishable key encapsulation mechanism, PRG as secure pseudorandom generator, and EO be a correct and satisfy statistical rerandomized ciphertext indistinguishability, the (semi-honest) ciphertext indistinguishability for the evaluation key and secret key owner and ciphertext well-formedness.*

Then, the protocol in Fig. 6 securely realizes ideal functionality in Fig. 2 for the join defined in Def. 4 for semi-honest corruption of one of the three parties C, D, S, and any amount of parties P_1 to P_T. In case of a corruption of D, the leakage graph of Def. 11 is leaked.

4 Matching Strategy

Recall from Fig. 1 that the view of each party for a specific record may be different and a record may have multiple identifiers (e.g., email address, phone). When combining datasets from multiple delegators, the uniqueness of the identifiers cannot be guaranteed as the same record might appear in more than one dataset. Thus, potential matches for each row can occur based on different identifiers across different delegators. For instance, a match on the jth identifier of record c_i may occur for keys in different positions between different parties (e.g., with $\mathsf{p}_{t,i',j'}$ with $i \neq i'$ and $j \neq j'$). Parties C and D in our protocols compute the left join as described in Def. 3 and acquire $\mathcal{J}_C$ and $\mathcal{J}_D$, respectively. To capture all the aforementioned matches, for T delegators, $\mathcal{J}_C$ and $\mathcal{J}_D$ have T permuted columns of secret shares which either correspond to shares of the associated metadata of one of the input parties (if a match was found) or to shares of `NULL` (in case no match was found).

As the number of delegators P_1 to P_T grows, it is natural for our resulting $\mathcal{J}_C$ and $\mathcal{J}_D$ tables to contain multiple secret shares of `NULL`. This becomes more evident if each individual dataset KV_t is relatively small compared to KV_C; even if all the records of KV_t match with records in KV_C, there would still be multiple unmatched records in KV_C which will get secret shares of `NULL`. To optimize both our matching and our downstream computation, we now delve into a matching strategy to generate one-to-many connections that do not depend on T and minimize the number of `NULL` secret shares.

First, C and D agree on a maximum number of connections K to capture. D performs a ranked left join by starting from the identifier with the highest priority in KV_C and checking whether it appears in each KV_t before moving to the next record in KV_C. After searching by the first key of each record in KV_C, D continues with the next identifier, and so on. If a record from P_t is matched, we mark that record as done and continue to the next record in order to avoid counting the same associated values more than once. For each record c_i, if K or more matches are found, D creates secret shares of the associated data of the first K records, otherwise (if less than K matches are found), D pads the remaining columns (up to K) with secret shares of `NULL`.

We note that this is an implementation-specific detail that can be trivially extended to different matching strategies. Each of the resulting tables $\mathcal{J}_C$ and $\mathcal{J}_D$ has K columns and captures a one-to-K matches for each record in the left join.

5 Real-World Applications

Recall that our ideal functionality $\mathcal{F}_{\mathsf{DPMC}}$ (and $\mathcal{F}_{\mathsf{DPMC}}$) consists of $\mathcal{F}_{\mathsf{JOIN}}$ and $\mathcal{F}_{\mathsf{CMP}}$. Our delegated protocols realize $\mathcal{F}_{\mathsf{JOIN}}$ and output secret shares to parties C and D for the left join of parties C and $P_1, \ldots, P_T$. Next, $\mathcal{F}_{\mathsf{CMP}}$ can be realized by running any general-purpose MPC between C and D. We foresee multiple real-world applications for $\mathcal{F}_{\mathsf{CMP}}$ that may leverage our architecture merging multiple private datasets across distrusting parties with a centralized entity (party C) to securely compute analytics. For instance, DPMC enables calculating the risk of a health condition by merging information held by a larger healthcare provider with data stored on millions of individual smart devices. In another example, an ad publisher holding user-provided information can measure advertising efficacy and offer personalization by merging with data held by multiple advertisers while still preserving user privacy. In this section, we focus on the latter and outline how DPMC enables privacy-preserving ad measurement and delivery of personalized advertising leveraging privacy-preserving machine learning. The former provides advertisers useful insights about how their ad campaigns are performing, while the latter enables delivering personalized ads while preserving user privacy.

5.1 Privacy-Preserving Ad Attribution

Inputs. We assume the following input data held by an ad publisher, denoted by C and T advertisers, denoted by $P_1, \ldots, P_T$.

- *Party* C is a company that holds a dataset of ad actions (i.e., clicks) performed by individuals on product-related advertisements. These ads were shown to users after they expressed an intent via an online search engine. Users may be shown ads related to multiple products owned by hundreds of advertisers.
- Advertisers P_1 to P_T, hold conversion information for their customers, such as purchase amount and time of the purchase.
- All parties (C, $P_1, \ldots, P_T$) also hold annotated sets of common identifiers (e.g., email addresses and phone numbers).

$\mathcal{F}_{\mathsf{JOIN}}$ phase. Executing the DPMC protocol for $\mathcal{F}_{\mathsf{JOIN}}$ with the above input data from C and multiple P parties, the following output is available at the ad publisher C and the delegate servers.

- *Party* C holds a mapping of secret shares of conversion data to a dataset of ad actions. This mapping does not reveal any new information to C apart from random-looking secret shares. In the case of no matches, party C receives secret shares of zero.
- *Party* D receives a set of secret shares of the conversion data or a dummy value (e.g., zero) that is also aligned to party's C records (i.e., left join). D gains insights into pairwise intersection sizes (in DPMC) or the intersection size of C with the union of all advertisers' sets (in $\mathsf{D}_s\mathsf{PMC}$). For example, when users have unique phone numbers and email addresses, in $\mathsf{D}_s\mathsf{PMC}$ D learns the intersection sizes of records where at least phone number, email address, or both matched between the company and the union of all advertisers' data. In a real-world scenario where D is a privacy-conscious non-profit organization, this level of leakage has fairly low privacy implications. If the uniqueness of identifiers cannot be assumed, the sizes of groups with the same identifiers are leaked.
- Parties P_1 to P_T, receive nothing.

$\mathcal{F}_{\mathsf{CMP}}$ phase. Parties C and D now hold secret shares of conversion metadata such as conversion time and values. C can then further input metadata of ad actions, such as click timestamp, as secret shares using the link to the original records that were established by the DPMC protocol. Now, parties C and D engage in multi-party computation to compute the attribution function that flags when a conversion (product was

bought) occurred within a pre-specified time window from the ad action. Note that the MPC computation is embarrassingly parallel given the row-wise output structure of DPMC. The output of the privacy-preserving ad attribution remains at the ad action level, hence remains secret shared between parties C and D and is used as an input into further downstream computations such as private measurement or personalization, described next.

5.2 Privacy-Preserving Analytics

Measuring the efficacy of advertising first requires computing aggregated conversion outcomes such as the total number of attributed conversions per campaign. Note that DPMC maintains the left join of the ad actions without revealing any user-level information to party C at any stage. Party C may attach campaign-level identifiers with limited entropy ensuring sufficient K-anonymity guarantees. At this point, parties C and D engage in another round of MPC (i.e., a new $\mathcal{F}_{\mathsf{CMP}}$ phase) to compute aggregated conversion outcomes per campaign. Finally, differentially private noise can be added to the aggregated outcomes within MPC before revealing the results to party C, so that C only learns noisy aggregates for each ad campaign.

5.3 Privacy-Preserving Personalization

Privacy-preserving personalization typically entails training a model to be able to estimate the relevance of potential ads for users. Note that privacy-preserving ad attribution during the data pre-processing phase generates secret shares of ad attribution outcomes for both parties C and D. Leveraging the mapping produced by DPMC from secret shares to original ad actions, party C may attach any private features to the private attribution outcomes without revealing any individually identifiable information. At this stage, parties C and D can run a new $\mathcal{F}_{\mathsf{CMP}}$ in multi-party computation for model training with privately input features (from party C) and secret shared labels (from both parties C and D). For example, CrypTen [33], a multi-party computation framework for machine learning, may be leveraged between the parties C and D downstream to the DPMC protocol. Similarly to the aforementioned analytics example, privacy-preserving personalization would also include differential privacy guarantees and we point avid readers to one such implementation [56].

6 Evaluations

Implementation & Setup. We implemented our protocols in Rust (1.62) and used the Dalek library for Elliptic Curve Cryptography with Ristretto for Curve25519 [18,28].[5] This enables the use of a fast curve while avoiding high-cofactor vulnerabilities. For symmetric encryption, we use the Fernet library with AES-128 in CBC mode, for public key encryption we use ElGamal with elliptic curves, and for the key encapsulation mechanism, we use ElGamal KEM.

We created artificial datasets where each record has one 128-bit identifier and two 64-bit associated values. The performance measurements were carried out on AWS m5.12xlarge EC2 instances (Intel Xeon at 3.1GHz, 48 vCPU, 192GB RAM). To simulate C, D, and multiple P parties we leverage three separate EC2 instances in the same region, where C and D are hosted by two separate instances, and the third instance hosts all parties P_1 to P_T. For our WAN experiments, we used three m5.12xlarge EC2 instances in N. Virginia, Ohio, and N. California. All parties communicate via RPC over TLS v1.3 using Protocol Buffers.

Varying number of delegators. In Fig. 7, we fixed the size of KV_C to 1 million and varied both the number of delegators and their dataset sizes. In the orange and red trends, we used a single delegator for DPMC and $\mathsf{D}_s\mathsf{PMC}$ with dataset sizes indicated by the x-axis. In the blue and green trends, we split the dataset into ten delegators, where each party has 1/10 of the input size shown on the x-axis. Although the combined size of the dataset of the ten parties is the same, the local computation for each delegator is significantly less. In this case, the performance time for each delegator is about ten times faster than having a single P_1 party with a bigger dataset.

Protocol time for delegator. Next, we fixed the input of party C to 1 million with a single identifier per record and varied the size of the dataset of the delegator. In Fig. 8 we show the execution times for party

[5] Our protocols are open-source at https://github.com/facebookresearch/Private-ID.

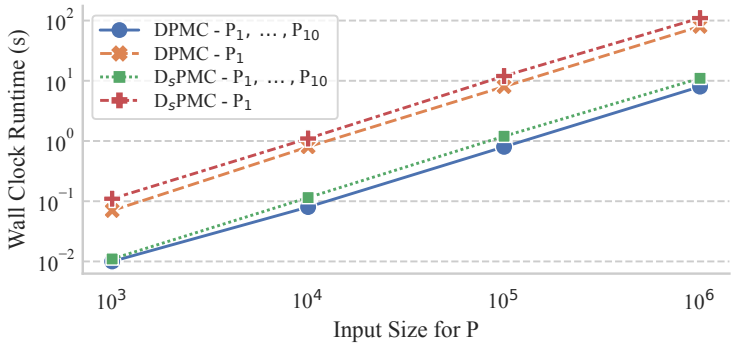


Fig. 7. Measured execution time of delegator for DPMC and D_sPMC with $m_C = 10^6$ and and increasing KV_P with intersection sizes of 50% of KV_P.

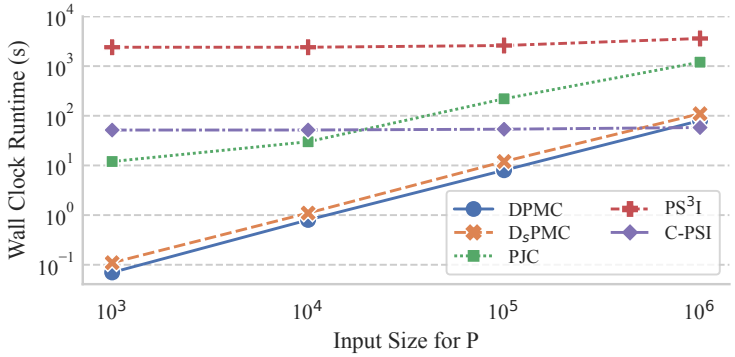


Fig. 8. Measured time of P for DPMC, D_sPMC, PJC, PS^{3}I, and Circuit-PSI with $m_C = 10^6$ and intersection sizes of 50% of m_t. All protocols are evaluated with a single delegator.

P for DPMC, D_sPMC, PJC, PS^{3}I, and Circuit-PSI. We use the PS^{3}I [10] and PJC [31] implementations from [9], which both use Paillier with a 2048-bit public key. Similarly to our protocols, both these protocols assume that party P has associated metadata: PS^{3}I generates additive secret shares, whereas PJC aggregates the associated values of the items in the intersection. Additionally, for fair comparisons, we implemented over-the-network communication between the sender and the receiver on the Circuit-PSI implementation of [50] (the implementation of [12] crashes with different dataset sizes). The blue and orange trends in Fig. 8 show the execution time of a single delegator running the DPMC and the D_sPMC protocols, respectively, which are approximately the same. We observe that the execution time for both is approximately 10× faster than party P in PJC and multiple orders of magnitude faster than PS^{3}I. The runtime in PJC scales linearly with P's dataset size, however, this is not the case with PS^{3}I as P's execution time is also affected by C's dataset. The runtime for Circuit-PSI is linear in the input of both parties, so it incurs high overheads when $m_C \gg m_P$. Both this and the previous experiments (Figs. 7 and 8) demonstrate the benefits of our delegated protocols for the delegator parties compared to the two-party protocols.

Varying intersection size. In our next experiment, we fixed the input size at 1 million records for both parties and the number of identifiers at 2 per record and varied the intersection size (1%, 25%, 50%, and 100%). We observed negligible performance variations (i.e., less than a second) for the different intersection sizes since our protocol always outputs the left join and depends on KV_C size.

Varying number of identifiers. We now show how the number of keys affects the performance of our protocols. We fixed the input size to 10^5 records for both parties and the intersection size to 50%. Fig. 9 shows the total time for DPMC (light green trend) and D_sPMC (light blue trend) as well as the input and output traffic for each party for DPMC. Notably, the communication of D_sPMC is similar for an increasing number of identifiers as the matching strategy is very similar for the two protocols.

Communication. In Table 2 we present the asymptotic costs (communication and number of exponentiations) of each protocol for each party. C and D incur similar communication overhead which scales with the size of

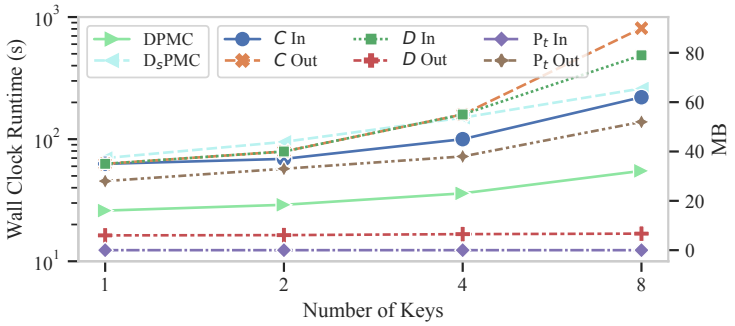


Fig. 9. Wall clock time for DPMC and D_sPMC. DPMC network traffic for each party C, D, and P_t with an increasing number of keys per row for $m_C = m_t = 10^5$ and an intersection size of 50% of m_C.

Table 2. Communication cost & number of exponentiation. T is the number of delegators; m_C and m_t are the set sizes of C and each delegator P_t, respectively, and $M \coloneqq \sum_{t=1}^{T} m_t$. $\mathcal{I} \coloneqq |\mathsf{KV}_C \cap \mathsf{KV}_P|$, where $\mathsf{KV}_P \coloneqq \{\mathsf{KV}_1 \cup \mathsf{KV}_2 \cup \ldots \cup \mathsf{KV}_T\}$.

	Party	C	D	S	P_t
DPMC	**Communication**	$\mathcal{O}(m_C + M)$	$\mathcal{O}(m_C + M)$	-	$\mathcal{O}(m_t)$
	Num. of Exp.	$2m_C + M$	$M + m_C - \mathcal{I}$	-	$2m_t + 1$
D_sPMC	**Communication**	$\mathcal{O}(m_C + M)$	$\mathcal{O}(m_C + M)$	$\mathcal{O}(M)$	$\mathcal{O}(m_t)$
	Num. of Exp.	$2m_C + 3M$	$M + m_C - \mathcal{I}$	$4M$	$2m_t$

KV_C and the delegators' datasets. The communication cost for each delegator P_t is linear to their dataset. In D_sPMC, party S incurs a linear communication to the size of all the delegators' datasets. Finally, we observe that the number of exponentiations of DPMC and D_sPMC are similar.

Table 3 shows each party's incoming and outgoing traffic in MBs. We observe a linear increase in the communication for each party as we increase the input sizes. Interestingly, we see that although D_sPMC performs more rounds than DPMC, the communication for each party is lower than DPMC. This happens because each P_t encrypts their `XOR` shares in order to prevent C from accessing them during the fourth step of the protocol. Finally, our protocols have similar communication as Circuit-PSI, which showed a linear increase with the dataset sizes. For reference, the outgoing communication for datasets of 10^6 elements was 424 MBs (344 from the sender and 80 from the receiver).

Table 3. For each party C, P, D, S we show In/Out in MB with $m_C = m_P$ and intersection size $\mathcal{I} = 50\%$ of m_C.

	Size	C [In/Out]	D [In/Out]	S [In/Out]	P_t [In/Out]
DPMC	10^3	0.3/0.3	0.3/0.1	-	0.1/0.3
	10^4	3.7/3.7	3.4/2.8	-	0.1/2.8
	10^5	33/33	33/4.8	-	0.1/28
	10^6	312/312	320/44	-	0.1/279
D_sPMC	10^3	0.2/0.2	0.1/0.1	0.1/0.1	0.1/0.1
	10^4	2.3/2.5	1.5/0.4	1/1	0.1/0.8
	10^5	22/24	14/4.3	9.5/9.5	0.1/8.5
	10^6	220/241	145/42	94/94	0.1/84.7

Two-party related works. We also compare our protocols with two-party works and vary the input size of each party from 10^3 to 10^6 while fixing the intersection size to 50%. As a baseline, we compare with multi-key Private-ID [8] which only focuses on private matching and does not consider associated data. To be in a similar setting, we run our protocols with a single party P. Fig. 10 shows how our delegated protocols significantly outperform both PS^{3}I and PJC by more than a factor of $10x$. On the other hand, our delegated protocols are only $\approx 1.8x$ slower than Private-ID although the latter does not include associated values and

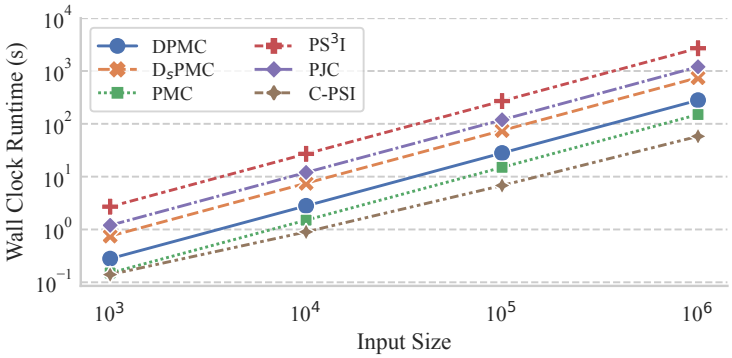


Fig. 10. Comparisons of DPMC and D_sPMC with two-party protocols: PJC, Private-ID (PMC), PJC, PS^{3}I, and Circuit-PSI with an increasing number of dataset sizes ($m_C = m_P$) and an intersection of 50%m_C. We use PMC as a baseline as it only performs matching and does not consider associated values.

is only between two parties. This means that our protocols process approximately twice the amount of data that Private-ID processes since for each row of KV_P, DPMC and D_sPMC also create secret shares of the associated data. Circuit-PSI is 3-4x times faster than our protocols but, as the other related works, only considers two parties who are both assumed to be online throughout the entire protocol execution and do not take into account any delegation methods.

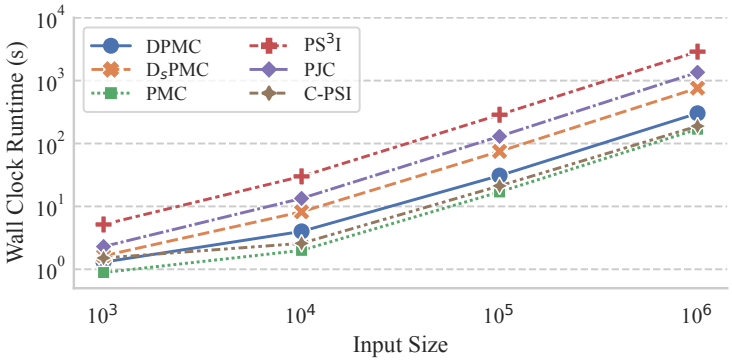


Fig. 11. Comparisons as in Fig. 10 over WAN.

In Fig. 11, we repeated the same experiments over WAN and observed a similar scaling for all the protocols. Interestingly, the margin between Circuit-PSI and our protocols became smaller as Circuit-PSI requires significantly more communication. Finally, note that these experiments do not offer a balanced assessment of our protocols as the benefits of the delegated setting are shown in Figs. 7 and 8. In the former, we observe that each delegator performs work proportional to their dataset size, while in the latter, the delegators have smaller datasets than C and they go offline after they outsource their datasets.

7 Concluding Remarks

We presented two delegated protocols that establish relations between datasets that are held by multiple distrusting parties and enable them to run any arbitrary secure computation. Our protocols allow the input parties to submit their records along with associated values and generate secret shares of the associated values for the matched records and secret shares of `NULL` otherwise. Notably, they facilitate the delegation of both the matching process and downstream secure computation to delegate parties. In contrast with prior works that only support two parties, our work is designed to scale to multiple input parties.

In addition, our delegated protocols enable one of the input parties to provide more data after the matching has been established which can be used for the downstream computation without requiring

rerunning the private matching process. We further introduced a rerandomizable encrypted OPRF (EO) primitive that extends beyond the classic two-party OPRF setting and allows multiple input providers to interact with an output receiver and a server and perform oblivious PRF evaluations. While prior works mostly focused on intersection and union, we focused on left-join matching and we demonstrated its benefits in privacy-preserving online advertising by performing private ad attribution measurement, privacy-preserving analytics, and personalization. Finally, our implementation demonstrates the efficiency of our constructions by outperforming related works.

Acknowledgments

The authors would like to thank Anderson Nascimento, Erik Taubeneck, Gaven Watson, Sanjay Saravanan, Shripad Gade, Pratik Sarkar, and Charles Gouert for the fruitful discussions and the anonymous reviewers for their feedback. The third author was partially supported by NSF awards #2101052, #2200161, #2115075, and ARPA-H SP4701-23-C-0074.

References

1. Toshinori Araki, Jun Furukawa, Yehuda Lindell, Ariel Nof, and Kazuma Ohara. High-throughput semi-honest secure three-party computation with an honest majority. In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *ACM CCS 2016: 23rd Conference on Computer and Communications Security*, pages 805–817, Vienna, Austria, October 24–28, 2016. ACM Press.
2. Amos Beimel. Secret-sharing schemes: A survey. In *International Conference on Coding and Cryptology*, pages 11–46, Berlin, Heidelberg, 2011. Springer.
3. Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In Dorothy E. Denning, Raymond Pyle, Ravi Ganesan, Ravi S. Sandhu, and Victoria Ashby, editors, *ACM CCS 93: 1st Conference on Computer and Communications Security*, pages 62–73, Fairfax, Virginia, USA, November 3–5, 1993. ACM Press.
4. Abhishek Bhowmick, Dan Boneh, Steve Myers, Kunal Talwar, and Karl Tarbe. The Apple PSI system, 2021.
5. Erik-Oliver Blass and Florian Kerschbaum. Private collaborative data cleaning via non-equi psi. In *2023 IEEE Symposium on Security and Privacy*, pages 1419–1434, San Francisco, CA, USA, May 2023. IEEE Computer Society Press.
6. Dan Bogdanov, Sven Laur, and Jan Willemson. Sharemind: A framework for fast privacy-preserving computations. In Sushil Jajodia and Javier López, editors, *ESORICS 2008: 13th European Symposium on Research in Computer Security*, volume 5283 of *Lecture Notes in Computer Science*, pages 192–206, Málaga, Spain, October 6–8, 2008. Springer, Heidelberg, Germany.
7. Dan Boneh, Elette Boyle, Henry Corrigan-Gibbs, Niv Gilboa, and Yuval Ishai. Lightweight techniques for private heavy hitters. In *2021 IEEE Symposium on Security and Privacy*, pages 762–776, San Francisco, CA, USA, May 24–27, 2021. IEEE Computer Society Press.
8. Prasad Buddhavarapu, Benjamin M Case, Logan Gore, Andrew Knox, Payman Mohassel, Shubho Sengupta, Erik Taubeneck, and Min Xue. Multi-key private matching for compute. Cryptology ePrint Archive, Report 2021/770, 2021. https://eprint.iacr.org/2021/770.
9. Prasad Buddhavarapu, Andrew Knox, Payman Mohassel, Shubho Sengupta, Erik Taubeneck, and Vlad Vlaskin. Private-ID. https://github.com/facebookresearch/Private-ID, 2020.
10. Prasad Buddhavarapu, Andrew Knox, Payman Mohassel, Shubho Sengupta, Erik Taubeneck, and Vlad Vlaskin. Private matching for compute. Cryptology ePrint Archive, Report 2020/599, 2020. https://eprint.iacr.org/2020/599.
11. Sílvia Casacuberta, Julia Hesse, and Anja Lehmann. SoK: Oblivious pseudorandom functions. Cryptology ePrint Archive, Report 2022/302, 2022. https://eprint.iacr.org/2022/302.
12. Nishanth Chandran, Divya Gupta, and Akash Shah. Circuit-PSI with linear complexity via relaxed batch OPPRF. *Proceedings on Privacy Enhancing Technologies*, 2022(1):353–372, January 2022.
13. Hao Chen, Zhicong Huang, Kim Laine, and Peter Rindal. Labeled PSI from fully homomorphic encryption with malicious security. In David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang, editors, *ACM CCS 2018: 25th Conference on Computer and Communications Security*, pages 1223–1237, Toronto, ON, Canada, October 15–19, 2018. ACM Press.
14. Michele Ciampi and Claudio Orlandi. Combining private set-intersection with secure two-party computation. In Dario Catalano and Roberto De Prisco, editors, *SCN 18: 11th International Conference on Security in Communication Networks*, volume 11035 of *Lecture Notes in Computer Science*, pages 464–482, Amalfi, Italy, September 5–7, 2018. Springer, Heidelberg, Germany.

15. Henry Corrigan-Gibbs and Dan Boneh. Prio: Private, robust, and scalable computation of aggregate statistics. In *Proceedings of the 14th USENIX Conference on Networked Systems Design and Implementation*, NSDI'17, page 259–282, USA, 2017. USENIX Association.
16. Ronald Cramer, Ivan Damgård, and Yuval Ishai. Share conversion, pseudorandom secret-sharing and applications to secure computation. In Joe Kilian, editor, *TCC 2005: 2nd Theory of Cryptography Conference*, volume 3378 of *Lecture Notes in Computer Science*, pages 342–362, Cambridge, MA, USA, February 10–12, 2005. Springer, Heidelberg, Germany.
17. Morten Dahl, Jason Mancuso, Yann Dupis, Ben Decoste, Morgan Giraud, Ian Livingstone, Justin Patriquin, and Gavin Uhma. Private Machine Learning in TensorFlow using Secure Computation. *CoRR*, abs/1810.08130:1–6, 2018.
18. Dalek-Cryptography. Dalek library for elliptic curve cryptography. GitHub, 2020. https://github.com/dalek-cryptography/curve25519-dalek.
19. Ivan Damgård and Yuval Ishai. Scalable secure multiparty computation. In Cynthia Dwork, editor, *Advances in Cryptology – CRYPTO 2006*, volume 4117 of *Lecture Notes in Computer Science*, pages 501–520, Santa Barbara, CA, USA, August 20–24, 2006. Springer, Heidelberg, Germany.
20. Hannah Davis, Christopher Patton, Mike Rosulek, and Phillipp Schoppmann. Verifiable Distributed Aggregation Functions. *Proceedings on Privacy Enhancing Technologies*, 2023(4):1–20, July 2023.
21. Daniel Demmler, Thomas Schneider, and Michael Zohner. ABY - A framework for efficient mixed-protocol secure two-party computation. In *ISOC Network and Distributed System Security Symposium – NDSS 2015*, pages 1–15, San Diego, CA, USA, February 8–11, 2015. The Internet Society.
22. Whitfield Diffie and Martin E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22(6):644–654, 1976.
23. Yevgeniy Dodis and Aleksandr Yampolskiy. A verifiable random function with short proofs and keys. In Serge Vaudenay, editor, *PKC 2005: 8th International Workshop on Theory and Practice in Public Key Cryptography*, volume 3386 of *Lecture Notes in Computer Science*, pages 416–431, Les Diablerets, Switzerland, January 23–26, 2005. Springer, Heidelberg, Germany.
24. Changyu Dong, Liqun Chen, and Zikai Wen. When private set intersection meets big data: an efficient and scalable protocol. In Ahmad-Reza Sadeghi, Virgil D. Gligor, and Moti Yung, editors, *ACM CCS 2013: 20th Conference on Computer and Communications Security*, pages 789–800, Berlin, Germany, November 4–8, 2013. ACM Press.
25. Thai Duong, Duong Hieu Phan, and Ni Trieu. Catalic: Delegated PSI cardinality with applications to contact tracing. In Shiho Moriai and Huaxiong Wang, editors, *Advances in Cryptology – ASIACRYPT 2020, Part III*, volume 12493 of *Lecture Notes in Computer Science*, pages 870–899, Daejeon, South Korea, December 7–11, 2020. Springer, Heidelberg, Germany.
26. Thanos Giannopoulos and Dimitris Mouris. Privacy preserving medical data analytics using secure multi party computation. an end-to-end use case. Master's thesis, National and Kapodistrian University of Athens, 2018.
27. Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game or A completeness theorem for protocols with honest majority. In Alfred Aho, editor, *19th Annual ACM Symposium on Theory of Computing*, pages 218–229, New York City, NY, USA, May 25–27, 1987. ACM Press.
28. Mike Hamburg et al. Ristretto, 2020. https://ristretto.group.
29. Marcella Hastings, Brett Hemenway, Daniel Noble, and Steve Zdancewic. SoK: General purpose compilers for secure multi-party computation. In *2019 IEEE Symposium on Security and Privacy*, pages 1220–1237, San Francisco, CA, USA, May 19–23, 2019. IEEE Computer Society Press.
30. Robert Helmer, Anthony Miyaguchi, and Eric Rescorla. Testing Privacy-Preserving Telemetry with Prio. https://hacks.mozilla.org/2018/10/testing-privacy-preserving-telemetry-with-prio, 2018.
31. Mihaela Ion, Ben Kreuter, Ahmet Erhan Nergiz, Sarvar Patel, Shobhit Saxena, Karn Seth, Mariana Raykova, David Shanahan, and Moti Yung. On deploying secure computing: Private intersection-sum-with-cardinality. In *EuroS&P*, pages 370–389, Genoa, Italy, 2020. IEEE.
32. Marcel Keller. MP-SPDZ: A versatile framework for multi-party computation. In Jay Ligatti, Xinming Ou, Jonathan Katz, and Giovanni Vigna, editors, *ACM CCS 2020: 27th Conference on Computer and Communications Security*, pages 1575–1590, Virtual Event, USA, November 9–13, 2020. ACM Press.
33. Brian Knott, Shobha Venkataraman, Awni Hannun, Shubho Sengupta, Mark Ibrahim, and Laurens van der Maaten. CrypTen: Secure Multi-Party Computation Meets Machine Learning. *Advances in Neural Information Processing Systems*, 34, 2021.
34. Vladimir Kolesnikov, Ranjit Kumaresan, Mike Rosulek, and Ni Trieu. Efficient batched oblivious PRF with applications to private set intersection. In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *ACM CCS 2016: 23rd Conference on Computer and Communications Security*, pages 818–829, Vienna, Austria, October 24–28, 2016. ACM Press.
35. Anja Lehmann. ScrambleDB: Oblivious (chameleon) pseudonymization-as-a-service. *Proceedings on Privacy Enhancing Technologies*, 2019(3):289–309, July 2019.

36. Tancrède Lepoint, Sarvar Patel, Mariana Raykova, Karn Seth, and Ni Trieu. Private join and compute from PIR with default. In Mehdi Tibouchi and Huaxiong Wang, editors, *Advances in Cryptology – ASIACRYPT 2021, Part II*, volume 13091 of *Lecture Notes in Computer Science*, pages 605–634, Singapore, December 6–10, 2021. Springer, Heidelberg, Germany.
37. Yehuda Lindell and Benny Pinkas. A proof of security of Yao's protocol for two-party computation. *Journal of Cryptology*, 22(2):161–188, April 2009.
38. Catherine Meadows. A more efficient cryptographic matchmaking protocol for use in the absence of a continuously available third party. In *1986 IEEE Symposium on Security and Privacy*, pages 134–134, Oakland, CA, USA, 1986. IEEE.
39. Peihan Miao, Sarvar Patel, Mariana Raykova, Karn Seth, and Moti Yung. Two-sided malicious security for private intersection-sum with cardinality. In Daniele Micciancio and Thomas Ristenpart, editors, *Advances in Cryptology – CRYPTO 2020, Part III*, volume 12172 of *Lecture Notes in Computer Science*, pages 3–33, Santa Barbara, CA, USA, August 17–21, 2020. Springer, Heidelberg, Germany.
40. Payman Mohassel, Peter Rindal, and Mike Rosulek. Fast database joins and PSI for secret shared data. In Jay Ligatti, Xinming Ou, Jonathan Katz, and Giovanni Vigna, editors, *ACM CCS 2020: 27th Conference on Computer and Communications Security*, pages 1271–1287, Virtual Event, USA, November 9–13, 2020. ACM Press.
41. Dimitris Mouris, Pratik Sarkar, and Nektarios Georgios Tsoutsos. PLASMA: Private, Lightweight Aggregated Statistics against Malicious Adversaries. Cryptology ePrint Archive, Report 2023/080, 2023. https://eprint.iacr.org/2023/080.
42. Dimitris Mouris and Nektarios Georgios Tsoutsos. Masquerade: Verifiable multi-party aggregation with secure multiplicative commitments. Cryptology ePrint Archive, Report 2021/1370, 2021. https://eprint.iacr.org/2021/1370.
43. Mahnush Movahedi, Benjamin M. Case, James Honaker, Andrew Knox, Li Li, Yiming Paul Li, Sanjay Saravanan, Shubho Sengupta, and Erik Taubeneck. Privacy-preserving randomized controlled trials: A protocol for industry scale deployment. In *Proceedings of the 2021 on Cloud Computing Security Workshop*, CCSW '21, page 59–69, New York, NY, USA, 2021. Association for Computing Machinery.
44. Muhammad Naveed, Seny Kamara, and Charles V. Wright. Inference attacks on property-preserving encrypted databases. In Indrajit Ray, Ninghui Li, and Christopher Kruegel, editors, *ACM CCS 2015: 22nd Conference on Computer and Communications Security*, pages 644–655, Denver, CO, USA, October 12–16, 2015. ACM Press.
45. Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In Jacques Stern, editor, *Advances in Cryptology – EUROCRYPT'99*, volume 1592 of *Lecture Notes in Computer Science*, pages 223–238, Prague, Czech Republic, May 2–6, 1999. Springer, Heidelberg, Germany.
46. Benny Pinkas, Mike Rosulek, Ni Trieu, and Avishay Yanai. SpOT-light: Lightweight private set intersection from sparse OT extension. In Alexandra Boldyreva and Daniele Micciancio, editors, *Advances in Cryptology – CRYPTO 2019, Part III*, volume 11694 of *Lecture Notes in Computer Science*, pages 401–431, Santa Barbara, CA, USA, August 18–22, 2019. Springer, Heidelberg, Germany.
47. Benny Pinkas, Thomas Schneider, Oleksandr Tkachenko, and Avishay Yanai. Efficient circuit-based PSI with linear communication. In Yuval Ishai and Vincent Rijmen, editors, *Advances in Cryptology – EUROCRYPT 2019, Part III*, volume 11478 of *Lecture Notes in Computer Science*, pages 122–153, Darmstadt, Germany, May 19–23, 2019. Springer, Heidelberg, Germany.
48. Benny Pinkas, Thomas Schneider, Christian Weinert, and Udi Wieder. Efficient circuit-based PSI via cuckoo hashing. In Jesper Buus Nielsen and Vincent Rijmen, editors, *Advances in Cryptology – EUROCRYPT 2018, Part III*, volume 10822 of *Lecture Notes in Computer Science*, pages 125–157, Tel Aviv, Israel, April 29 – May 3, 2018. Springer, Heidelberg, Germany.
49. Benny Pinkas, Thomas Schneider, and Michael Zohner. Faster private set intersection based on OT extension. In Kevin Fu and Jaeyeon Jung, editors, *USENIX Security 2014: 23rd USENIX Security Symposium*, pages 797–812, San Diego, CA, USA, August 20–22, 2014. USENIX Association.
50. Peter Rindal and Phillipp Schoppmann. VOLE-PSI: Fast OPRF and circuit-PSI from vector-OLE. In Anne Canteaut and Franccois-Xavier Standaert, editors, *Advances in Cryptology – EUROCRYPT 2021, Part II*, volume 12697 of *Lecture Notes in Computer Science*, pages 901–930, Zagreb, Croatia, October 17–21, 2021. Springer, Heidelberg, Germany.
51. Mike Rosulek and Ni Trieu. Compact and malicious private set intersection for small sets. In Giovanni Vigna and Elaine Shi, editors, *ACM CCS 2021: 28th Conference on Computer and Communications Security*, pages 1166–1181, Virtual Event, Republic of Korea, November 15–19, 2021. ACM Press.
52. Adi Shamir. How to share a secret. *Communications of the Association for Computing Machinery*, 22(11):612–613, November 1979.
53. Silvia Vermicelli, Livio Cricelli, and Michele Grimaldi. How can crowdsourcing help tackle the covid-19 pandemic? an explorative overview of innovative collaborative practices. *R&D Management*, 51(2):183–194, 2021.
54. Xiao Wang, Alex J. Malozemoff, and Jonathan Katz. EMP-toolkit: Efficient MultiParty computation toolkit. https://github.com/emp-toolkit, 2016.

55. Andrew Chi-Chih Yao. Protocols for secure computations (extended abstract). In *23rd Annual Symposium on Foundations of Computer Science*, pages 160–164, Chicago, Illinois, November 3–5, 1982. IEEE Computer Society Press.
56. Sen Yuan, Milan Shen, Ilya Mironov, and Anderson C. A. Nascimento. Practical, label private deep learning training based on secure multiparty computation and differential privacy. *IACR Cryptol. ePrint Arch.*, 1:835, 2021.

A Additional Definitions

Below, we provide Algs. 1 and 2 for Defs. 3 and 4. For simplicity, our DPMC protocol in Fig. 4 uses single keys. Alg. 1 computes the join as outlined in Def. 3 for multiple keys but it can be easily adjusted for single keys. Alg. 2 computes the join as outlined in Def. 4.

Algorithm 1 Join for DPMC (Fig. 4 and Def. 3).

Match on: $\{\mathsf{hc}_{\hat{t},i,j}\}_{j\in[m_{\hat{t},i}],i\in[m_{\hat{t}}],\hat{t}\in[T]}\}$ and $\{(\mathsf{h}_{C,i,j})_{j\in[m_{C,i}]}\}_{i\in[m_C]}$

$\mathcal{J} \coloneqq \emptyset$ ▷ Initialize join.
For $i \in [m_C]$, $\hat{t} \in [T]$: ▷ Perform the join.
  For $j \in [m_{C,i}]$: ▷ Set of matched indices.
    $\mathsf{S}_{i,j,\hat{t}} \coloneqq \{i' \in [m_{\hat{t}}] \mid \exists j' \in [m_{\hat{t},i'}] \text{ s.t. } \mathsf{hc}_{\hat{t},i',j'} = \mathsf{h}_{C,i,j}\}$
  If $\bigcup_j \mathsf{S}_{i,j,\hat{t}} \neq \emptyset$: ▷ If a match was found.
    $j_{i,\hat{t}} \coloneqq \min(j \in [m_{C,i}] \text{ s.t. } \mathsf{S}_{i,j,\hat{t}} \neq \emptyset)$
    Pick $i' \in \mathsf{S}_{i,j_{i,\hat{t}},\hat{t}}$ ▷ i' is unique for each i.
    Add $(i', \hat{t})$ to $\mathcal{J}$.

Algorithm 2 Join for D_sPMC (Fig. 6 and Def. 4).

Match on: $\{\mathsf{h}_{i,j}\}_{i\in[M],j\in[m_i]}\}$ and $\{\mathsf{h}_{C,i,j}\}_{i\in[m_C],j\in[m_{C,i}]}$

$\mathcal{J} \coloneqq \emptyset$ ▷ Initialize join.
For $i \in [m_C]$: ▷ Perform the join.
  For $j \in [m_{C,i}]$: ▷ For each column.
    $\mathsf{S}_{i,j} \coloneqq \{i' \in [M] \mid \exists j' \in [m_{i'}] \text{ s.t. } \mathsf{h}_{i',j'} = \mathsf{h}_{C,i,j}\}$
  $t_i \coloneqq 1$ ▷ Keep track of number of matches for row i.
  $\mathsf{S}_T \coloneqq \emptyset$
  For $j \in [m_{C,i}]$: ▷ For each column.
    If $\bigcup_{j\in[m_{C,i}]} \mathsf{S}_{i,j} \setminus \mathsf{S}_T \neq \emptyset$ **and** $t_i < T$: ▷ If a match was found.
      $i' \xleftarrow{\mathrm{R}} \mathsf{S}_{i,j} \setminus \mathsf{S}_T$
      $\mathsf{S}_T \coloneqq \mathsf{S}_T \cup \{i'\}$
      $t_i \coloneqq t_i + 1$ ▷ t_i matches for row i.
      Add (i', t_i) to $\mathcal{J}$.

Definition 12 (DDH Assumption). *[22] Let $\mathbb{G}(\kappa)$ be a group parameterized by security parameter κ and g be a generator. We say that the Decisional Diffie–Hellman (DDH) assumption holds in group $\mathbb{G}(\kappa)$ if for every* ppt *adversary $\mathcal{A}$:*

$$\left|\Pr[\mathcal{A}(g, g^a, g^b, g^{ab}) = 1] - \Pr[\mathcal{A}(g, g^a, g^b, g^c) = 1]\right| \leq \mathsf{negl},$$

where the probability is taken over $a \xleftarrow{\mathrm{R}} \mathbb{Z}_q$, $b \xleftarrow{\mathrm{R}} \mathbb{Z}_q$, $c \xleftarrow{\mathrm{R}} \mathbb{Z}_q$ and the random coins of $\mathcal{A}$.

Definition 13 (Pseudorandom Generator). *We call a deterministic polynomial time algorithm* PRG *a pseudorandom generator if for any* ppt *adversary $\mathcal{A}$,*

$$\left|\Pr[\mathcal{A}(x) = 1] - \Pr[\mathcal{A}(u) = 1]\right| \leq \mathsf{negl},$$

where $\ell > \kappa$, $u \xleftarrow{R} \{0,1\}^\ell$, $\mathsf{seed} \xleftarrow{R} \{0,1\}^\kappa$ *and* $x = \mathsf{PRG}(\mathsf{seed})$.

Definition 14 (Random Oracle). *[3] A random oracle* RO *is a family of functions that maps an input from* $\{0,1\}^*$ *to an* ℓ*-bit image* $\{0,1\}^\ell$ *s.t. each output is selected uniformly and independently.*

Definition 15 (Symmetric Key Encryption). *A symmetric encryption scheme parameterized with security parameter* κ *is a triplet of algorithms* $(\mathsf{SKE.KG}, \mathsf{SKE.Enc}, \mathsf{SKE.Dec})$ *with the following syntax.*

- $\mathsf{SKE.KG}(1^\kappa)$*: On input* 1^κ *output secret key* sk.
- $\mathsf{SKE.Enc}(\mathsf{sk}, x)$*: On input* (sk, x), SKE.Enc *outputs a ciphertext* ct.
- $\mathsf{SKE.Dec}(\mathsf{sk}, \mathsf{ct})$*: On input* $(\mathsf{sk}, \mathsf{ct})$, SKE.Dec *outputs a message* x.

For correctness, we ask that for any message $x \in \{0,1\}^*$,

$$\Pr_{\mathsf{sk} \leftarrow \mathsf{SKE.KG}(1^\kappa)}[\mathsf{SKE.Dec}(\mathsf{sk}, \mathsf{SKE.Enc}(\mathsf{sk}, x)) = x] \geq 1 - \mathsf{negl}.$$

Definition 16 (Public Key Encryption). *A public encryption scheme parameterized with security parameter* κ *is a triplet of algorithms* $(\mathsf{PKE.KG}, \mathsf{PKE.Enc}, \mathsf{PKE.Dec})$ *with the following syntax:*

- $\mathsf{PKE.KG}(1^\kappa)$*: On input* 1^κ *output a key pair* $(\mathsf{pk}, \mathsf{sk})$.
- $\mathsf{PKE.Enc}(\mathsf{pk}, x)$*: On input* (pk, x), PKE.Enc *outputs a ciphertext* ct.
- $\mathsf{PKE.Dec}(\mathsf{sk}, \mathsf{ct})$*: On input* $(\mathsf{sk}, \mathsf{ct})$, PKE.Dec *outputs a message* x.

For correctness, we ask that for any message $x \in \{0,1\}^*$,

$$\Pr_{(\mathsf{pk},\mathsf{sk}) \leftarrow \mathsf{PKE.KG}(1^\kappa)}[\mathsf{PKE.Dec}(\mathsf{sk}, \mathsf{PKE.Enc}(\mathsf{pk}, x)) = x] \geq 1 - \mathsf{negl}.$$

B Rerandomizable Encrypted OPRF (EO)

B.1 EO Definition

In Def. 10, we introduce a new construction called rerandomizable encrypted OPRF (EO) that allows two parties to encrypt, mask, and shuffle their data.

Definition 17 (Pseudorandomness of the Evaluation). *We say that the evaluation is pseudorandom if for any* ppt *adversary* $\mathcal{A}$ *with query access to* $\mathcal{O}_{\mathsf{Eval}(\mathsf{sk},\cdot)}$ *(*$\mathcal{O}_u(\cdot)$*),*

$$\left|\Pr[\mathcal{A}^{\mathcal{O}_{\mathsf{Eval}(\mathsf{sk},\cdot)}}(\mathsf{pk}, \mathsf{pf}) = 1] - \Pr[\mathcal{A}^{\mathcal{O}_u(\cdot)}(\mathsf{pk}, \mathsf{pf}) = 1]\right| \leq \mathsf{negl},$$

where $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KG}(1^\kappa)$, $(\mathsf{pf}, \mathsf{ek}) \leftarrow \mathsf{EKG}(1^\kappa)$*, and for* $x \in \{0,1\}^\kappa$, $\mathcal{O}_u$ *outputs a uniform* y *whereas* $\mathcal{O}_{\mathsf{Eval}(\mathsf{sk},\cdot)}$ *outputs* $y = \mathsf{Eval}(\mathsf{sk}, x)$.

A stronger definition of pseudorandomness of the evaluation is malicious pseudorandomness of the oblivious evaluation. We add the definition for completeness even though our construction only satisfies the pseudorandomness of the evaluation.

Definition 18 (Malicious Pseudorandomness of the Oblivious Evaluation). *We say that the oblivious evaluation is pseudorandom if for any* ppt *adversary* $\mathcal{A}$ *with query access to* $\mathcal{O}_{\mathsf{Dec}(\mathsf{sk},\mathsf{OEval}(\mathsf{ek},\cdot))}$ *(*$\mathcal{O}_u(\cdot)$*),*

$$\left|\Pr[\mathcal{A}^{\mathcal{O}_{\mathsf{Dec}(\mathsf{sk},\mathsf{OEval}(\mathsf{ek},\cdot))}}(\mathsf{pk}, \mathsf{pf}) = 1] - \Pr[\mathcal{A}^{\mathcal{O}_u(\cdot)}(\mathsf{pk}, \mathsf{pf}) = 1]\right| \leq \mathsf{negl},$$

where $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KG}(1^\kappa)$, $(\mathsf{pf}, \mathsf{ek}) \leftarrow \mathsf{EKG}(1^\kappa)$*, and for* $\mathsf{ct} \leftarrow \mathcal{A}^{\mathcal{O}}(\mathsf{pk}, \mathsf{pf})$ *with* $\mathsf{Dec}(\mathsf{sk}, \mathsf{OEval}(\mathsf{ek}, \mathsf{ct})) \neq \perp$, $\mathcal{O}_u$ *outputs a uniform* y *whereas* $\mathcal{O}_{\mathsf{Dec}(\mathsf{sk},\mathsf{OEval}(\mathsf{ek},\cdot))}$ *outputs* $y = \mathsf{Dec}(\mathsf{sk}, \mathsf{OEval}(\mathsf{ek}, \mathsf{ct}))$.

Definition 19 (Ciphertext Indistinguishability for Evaluation Key (ek) Owner). *We call EO ciphertext indistinguishable for the evaluation key owner if for any* ppt *algorithm* $\mathcal{A}$*,*

$$\left|\Pr[\mathcal{A}(\mathsf{pk}, \mathsf{ct}_0) = 1] - \Pr[\mathcal{A}(\mathsf{pk}, \mathsf{ct}_1) = 1]\right| \leq \mathsf{negl},$$

where $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KG}(1^\kappa)$*. In the adaptive malicious setting* $(m_0, m_1, \mathsf{pf}) \leftarrow \mathcal{A}(\mathsf{pk})$ *whereas in the semi-honest setting* $(\mathsf{pf}, \mathsf{ek}) \leftarrow \mathsf{EKG}(1^\kappa)$ *and* $(m_0, m_1) \leftarrow \mathcal{A}(\mathsf{pk}, \mathsf{pf}, \mathsf{ek})$*.* $\forall i \in \{0,1\} : \mathsf{ct}_i \leftarrow \mathsf{Enc}(\mathsf{pk}, \mathsf{pf}, x_i)$.

Definition 20 (Ciphertext Indistinguishability for Secret Key Owner). *We call EO ciphertext indistinguishable for the secret key owner if for any* ppt *algorithm* $\mathcal{A}$*,*

$$\left|\Pr[\mathcal{A}(\mathsf{pk}, \mathsf{ct}_0) = 1] - \Pr[\mathcal{A}(\mathsf{pk}, \mathsf{ct}_1) = 1]\right| \leq \mathsf{negl},$$

where $(\mathsf{pf}, \mathsf{ek}) \leftarrow \mathsf{KG}(1^\kappa)$*. In the adaptive malicious setting* $(m_0, m_1,\ \mathsf{pk}) \leftarrow \mathcal{A}(\mathsf{pf})$ *whereas in the semi-honest setting* $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KG}(1^\kappa)$ *and* $(m_0, m_1) \leftarrow \mathcal{A}(\mathsf{pk},\ \mathsf{pf}, \mathsf{sk})$*.* $\forall i \in \{0, 1\} : \mathsf{ct}_i \leftarrow \mathsf{Enc}(\mathsf{pk},\ \mathsf{pf}, m_i)$*.*

Definition 21 (Rerandomized Ciphertext Indistinguishability). *We call EO rerandomized ciphertext indistinguishable if for any* ppt *algorithm* $\mathcal{A}$*,*

$$\left|\Pr[\mathcal{A}(\mathsf{pk}, \mathsf{ct}_0) = 1] - \Pr[\mathcal{A}(\mathsf{pk}, \mathsf{ct}_1) = 1]\right| \leq \mathsf{negl},$$

$(x, \mathsf{pk}, \mathsf{pf}) \leftarrow \mathcal{A}(1^\kappa)$*,* $\mathsf{ct}_0 \leftarrow \mathsf{Rnd}(\mathsf{pk}, \mathsf{pf}, \mathsf{Enc}(\ \mathsf{pk}, \mathsf{pf}, x))$ *and* $\mathsf{ct}_1 \leftarrow \mathsf{Enc}(\ \mathsf{pk}, \mathsf{pf}, x)$*.*

Definition 22 (Ciphertext Well-Formedness). *We call an EO scheme ciphertext well-formed if for any* x_0, x_1 *with* $\mathsf{OEval}(\mathsf{ek}, x_0) = \mathsf{OEval}(\mathsf{ek}, x_1)$

$$\Delta_s(\mathsf{ct}_0, \mathsf{ct}_1) \leq \mathsf{negl},$$

where $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KG}(1^\kappa)$*,* $(\mathsf{pf}, \mathsf{ek}) \leftarrow \mathsf{EKG}(1^\kappa)$ *and* Δ_s *is the statistical distance.*

Definition 23 (Evaluated Ciphertext Simulatability). *We call an EO scheme evaluated ciphertext simulatable if there exists an* ppt *algorithm* EO.Sim *such that for any* x*,*

$$\Delta_s(\mathsf{ect}_0, \mathsf{ect}_1) \leq \mathsf{negl},$$

where $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KG}(1^\kappa)$*,* $(\mathsf{pf}, \mathsf{ek}) \leftarrow \mathsf{EKG}(1^\kappa)$*,* $\mathsf{ect}_0 \leftarrow \mathsf{OEval}(\mathsf{ek}, \mathsf{Enc}(\mathsf{pk}, \mathsf{pf}, x))$*,* $\mathsf{ect}_1 \leftarrow \mathsf{EO.Sim}(\mathsf{pk}, \mathsf{pf}, \mathsf{sk}, \mathsf{Eval}(\mathsf{ek}, x))$ *and* Δ_s *is the statistical distance.*

B.2 EO Construction and Security Analysis

In this section, we instantiate our EO construction in cyclic groups and prove its security against semi-honest adversaries.

Definition 24 (EO Construction in Cyclic Groups). *Let* g *be a generator of a cyclic group* $\mathbb{G}$ *with order* q *and* $H_{\mathbb{G}}(\cdot) : \{0, 1\}^* \to \mathbb{G}$ *a hash function. Then the EO collection of algorithms is constructed as follows.*

- $\mathsf{KG}(1^\kappa)$*: Sample* $a \xleftarrow{R} \mathbb{Z}_q$ *and output* $(\mathsf{pk} \coloneqq g^a, \mathsf{sk} \coloneqq a)$*.*
- $\mathsf{EKG}(1^\kappa)$*: Sample* $b \xleftarrow{R} \mathbb{Z}_q$ *and output* $(\mathsf{pf} \coloneqq g^b, \mathsf{ek} \coloneqq b)$*.*
- $\mathsf{Eval}(\mathsf{ek}, x)$*: Output* $y = H_{\mathbb{G}}(x)^{\mathsf{ek}}$*.*
- $\mathsf{Enc}(\mathsf{pk}, \mathsf{pf}, x)$*: Sample* $r \xleftarrow{R} \mathbb{Z}_q$ *and define* $\mathsf{ct}_1 \coloneqq \mathsf{pf}^r$*,* $\mathsf{ct}_2 \coloneqq \mathsf{pk}^r \cdot H_{\mathbb{G}}(x)$*. If* $\mathsf{pk} \neq \mathsf{pf}$ *output ciphertext* $\mathsf{ct} \coloneqq (\mathsf{ct}_1, \mathsf{ct}_2)$ *otherwise output* $\perp$*.*
- $\mathsf{Rnd}(\mathsf{pk}, \mathsf{pf}, \mathsf{ct})$*: Let* $\mathsf{ct} = (\mathsf{ct}_1, \mathsf{ct}_2)$*. Sample* $r \xleftarrow{R} \mathbb{Z}_q$ *and define* $\mathsf{ct}'_1 \coloneqq \mathsf{ct}_1 \cdot \mathsf{pf}^r$*,* $\mathsf{ct}'_2 \coloneqq \mathsf{ct}_2 \cdot \mathsf{pk}^r$ *and output ciphertext* $\mathsf{ct}' \coloneqq (\mathsf{ct}'_1,\ \mathsf{ct}'_2)$*.*
- $\mathsf{OEval}(\mathsf{ek}, \mathsf{ct})$*: Let* $\mathsf{ct} = (\mathsf{ct}_1, \mathsf{ct}_2)$*. Define* $\mathsf{ect}_2 \coloneqq \mathsf{ct}_2^{\mathsf{ek}}$ *and output* $\mathsf{ect} \coloneqq (\mathsf{ct}_1,\ \mathsf{ect}_2)$*.*
- $\mathsf{Dec}(\mathsf{sk}, \mathsf{ect})$*: Let* $\mathsf{ect} = (\mathsf{ect}_1, \mathsf{ect}_2)$*. Output* $y \coloneqq \mathsf{ect}_2 / \mathsf{ect}_1^{\mathsf{sk}}$*.*

For correctness, we ask that for any $x \in \{0, 1\}^*$*,*

$$\Pr[\mathsf{Dec}(\mathsf{sk}, \mathsf{OEval}(\mathsf{ek}, \mathsf{Rnd}(\mathsf{pk}, \mathsf{pf}, \mathsf{Enc}(\mathsf{pk}, \mathsf{pf}, x)))) = \mathsf{Eval}(\mathsf{ek}, x)] \geq 1 - \mathsf{negl},$$

where $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KG}(1^\kappa)$ *and* $(\mathsf{pf}, \mathsf{ek}) \leftarrow \mathsf{EKG}(1^\kappa)$*.*

Lemma 1. *Def. 24 defines a correct EO scheme.*

Proof. Let $(g^a, a) \leftarrow \mathsf{KG}(1^\kappa)$ and $(g^b, b) \leftarrow \mathsf{EKG}(1^\kappa)$. The correctness of the EO construction is satisfied as shown below:

$$\begin{aligned}
\mathsf{Dec}(a, \mathsf{OEval}(b, \mathsf{Rnd}(g^a, g^b, \mathsf{Enc}(g^a, g^b, x)))) &= \mathsf{Eval}(b, x) \Leftrightarrow \\
\mathsf{Dec}(a, \mathsf{OEval}(b, \mathsf{Rnd}(g^a, g^b, (g^{br}, g^{ar} \cdot H_{\mathbb{G}}(x))))) &= H_{\mathbb{G}}(x)^b \Leftrightarrow \\
\mathsf{Dec}(a, \mathsf{OEval}(b, (g^{br} \cdot g^{br'}, g^{ar} \cdot g^{ar'} \cdot H_{\mathbb{G}}(x)))) &= H_{\mathbb{G}}(x)^b \Leftrightarrow \\
\mathsf{Dec}(a, \mathsf{OEval}(b, (g^{b(r+r')}, g^{a(r+r')} \cdot H_{\mathbb{G}}(x)))) &= H_{\mathbb{G}}(x)^b \Leftrightarrow \\
\mathsf{Dec}(a, (g^{b(r+r')}, (g^{a(r+r')} \cdot H_{\mathbb{G}}(x))^b)) &= H_{\mathbb{G}}(x)^b \Leftrightarrow \\
\mathsf{Dec}(a, (g^{b(r+r')}, g^{ab(r+r')} \cdot H_{\mathbb{G}}(x)^b)) &= H_{\mathbb{G}}(x)^b \Leftrightarrow \\
g^{ab(r+r')} \cdot H_{\mathbb{G}}(x)^b / (g^{b(r+r')})^a &= H_{\mathbb{G}}(x)^b.
\end{aligned}$$

The construction is secure against semi-honest adversaries under the DDH assumption. The bottleneck that prevents malicious security is the OPRF $H(x)^k$. This OPRF only provides semi-honest security since a malicious delegator might send an arbitrary group element X instead of $H(x)$. In that case, it does not result in an OPRF since it satisfies linear relations, e.g., $X^k \cdot Y^k = (X \cdot Y)^k$.

We have outlined what is needed from the EO for malicious security in Defs. 18-20. The main bottleneck for our $H(x)^k$ based construction is Def. 18 (Defs. 19 and 20 seem to hold when making stronger assumptions than DDH). Other PRF candidates seem significantly less efficient (i.e., lowMC) or require stronger assumptions (e.g., Dodis-Yampolskiy PRF [23]). In Appendix C, we show that our EO primitive is compatible with the MPC shuffle protocol of [40] by relying on the EO rerandomization procedure.

Lemma 2. *Def. 24 satisfies pseudorandomness of the evaluation under the DDH assumption in the Random Oracle Model.*

Proof. We use a sequence of hybrids in which we replace step by step (based on the order of random oracle queries) $\mathsf{Eval}(\mathsf{ek}, x)$ with a uniform group element. If there is a distinguisher against the pseudorandomness of Eval with probability ϵ then there is a distinguisher against at least two consecutive intermediate hybrids with probability ϵ/Q, where Q is the maximum between the amount of random oracle and Eval oracle queries. Given such a distinguisher, we build a distinguisher against DDH as follows. The DDH distinguisher receives challenge A, B, C and sets $\mathsf{pf} := A$. Once the random oracle query is made that differentiates the two hybrids (let that be the i^*th query), it programs $H_{\mathbb{G}}(x) := B$. For all following queries $i > i^*$ program $H_{\mathbb{G}}(x) := g^{r_i}$, where $r_i \xleftarrow{R} \mathbb{Z}_q$. When a query for x to the Eval oracle is made, query x to the random oracle if it has not been made yet. If x matches the query i^*, respond with C. If x corresponds to a query $i < i^*$, respond with a uniform group element. Otherwise respond with B^{r_i}.

If $A = g^a, B = g^b, C = g^c$ then the DDH distinguisher simulates the first of the two hybrids. In case of uniform A, B, C it simulates the second of the two hybrids where the output of the Eval oracle that corresponds to the i^*th message is uniform.

Since Q is polynomial and the distinguishing probability against DDH is negligible, the probability to break the pseudorandomness of Eval is also negligible. □

Lemma 3. *Def. 24 is ciphertext indistinguishable for the evaluation key owner in the semi-honest setting under the DDH assumption.*

Proof. We use three hybrids, the first hybrid uses x_0 for the challenge ciphertext. In the second hybrid, the ciphertext is independent of the message. The third hybrid uses x_1 for the challenge ciphertext. We show now that these three hybrids cannot be distinguished based on the DDH assumption.

We build a DDH distinguisher for hybrid one and two (two and three) as follows. It receives DDH challenge A, B, C and samples $(\mathsf{pf}, \mathsf{ek}) \leftarrow \mathsf{EKG}(1^\kappa)$. It defines $\mathsf{pk} := A$ and sends $(\mathsf{pk}, \mathsf{ek}, \mathsf{pf})$ to the distinguisher against the ciphertext indistinguishability. It receives x_0 and x_1. Return challenge ciphertext $\mathsf{ct}_1 := B^{\mathsf{ek}}$, $\mathsf{ct}_2 := C \cdot x_0$ ($\mathsf{ct}_2 := C \cdot x_1$). Output the output of the ciphertext indistinguishability distinguisher.

If $A = g^a, B = g^b, C = g^c$ then the challenge ciphertext follows the output distribution of Enc for x_0 as in the first hybrid (and m_1 in the third hybrid). Otherwise, the challenge ciphertext is independent of the message as in the second hybrid. □

Lemma 4. *Def. 24 is ciphertext indistinguishable for the secret key owner in the semi-honest setting under the DDH assumption for prime groups (every element is a generator).*

Proof. We use three hybrids, the first hybrid uses x_0 for the challenge ciphertext. In the second hybrid, the ciphertext is independent of the message. The third hybrid uses x_1 for the challenge ciphertext. We show now that these three hybrids cannot be distinguished based on the DDH assumption.

We build a DDH distinguisher for hybrid one and two (two and three) as follows. It receives DDH challenge A, B, C and samples $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KG}(1^\kappa)$. It defines $\mathsf{pf} \coloneqq A$ and sends $(\mathsf{pk}, \mathsf{sk}, \mathsf{pf})$ to the distinguisher against the ciphertext indistinguishability. It receives x_0 and x_1. Return challenge ciphertext $\mathsf{ct}_1 \coloneqq C$, $\mathsf{ct}_2 \coloneqq B^{\mathsf{sk}} \cdot x_0$ ($\mathsf{ct}_2 \coloneqq B^{\mathsf{sk}} \cdot x_1$). Output the output of the ciphertext indistinguishability distinguisher.

If $A = g^a, B = g^b, C = g^c$ then the challenge ciphertext follows the output distribution of Enc for x_0 (x_1) as in the first hybrid (third hybrid). Otherwise, the challenge ciphertext is independent of the message as in the second hybrid as long as B is a generator of the group and thus B^{sk} is uniform for a uniform B. □

Lemma 5. *Def. 24 is statistically randomized ciphertext indistinguishable.*

Proof. Let $\mathsf{ct} \coloneqq (g^{br}, g^{ar} \cdot H_{\mathbb{G}}(x))$ be an encryption of x for some random $r \in \mathbb{Z}_q$. Then the randomized ciphertext $\mathsf{Rnd}(g^a, g^b, \mathsf{ct})$ is defined as $(g^{br} \cdot g^{br'}, g^{ar} \cdot g^{ar'} \cdot H_{\mathbb{G}}(x)) = (g^{b(r+r')}, g^{a(r+r')} \cdot H_{\mathbb{G}}(x))$ for random $r' \in \mathbb{Z}_q$. Since both r and r' are random elements in $\mathbb{Z}_q$, $r + r'$ is also a random element in $\mathbb{Z}_q$ and the ciphertext is statistically randomized ciphertext indistinguishable. □

Lemma 6. *Let* sk *and* q *be coprime. Then Def. 24 is ciphertext well formed.*

Proof. Ciphertext well-formedness demands that messages that result in the same PRF evaluation have an identical ciphertext distribution. In the construction of Def. 24 the ciphertext only depends on $H_{\mathbb{G}}(x)$ and the output of Eval is $H_{\mathbb{G}}(x)^{\mathsf{ek}}$. Now, let there be x_0 and x_1 with $H_{\mathbb{G}}(x_0)^{\mathsf{ek}} = H_{\mathbb{G}}(x_1)^{\mathsf{ek}}$ and let for $b \in \{0, 1\}$, $H_{\mathbb{G}}(x_b) = g^{r_b}$. Then $(r - r') \cdot \mathsf{ek} = 0 \mod q$ and therefore $(r - r') = 0$ such that $H_{\mathbb{G}}(x_0) = H_{\mathbb{G}}(x_1)$ and the ciphertexts have the same distribution or ek would divide the group order q and therefore not be coprime. □

Lemma 7. *Def. 24 is evaluated ciphertext simulatable.*

Proof. $\mathsf{EO.Sim}$ takes as input $\mathsf{pk} = g^a$, $\mathsf{pf} = g^b$, $\mathsf{sk} = a$ and $y = H_{\mathbb{G}}(x)^b$. It outputs $\mathsf{ect} = (\mathsf{ect}_0, \mathsf{ect}_1)$ where $\mathsf{ect}_0 = \mathsf{pf}^r$, $\mathsf{ect}_1 = \mathsf{pf}^{ar} \cdot y$. This identically distributed as $\mathsf{ect} = \mathsf{OEval}(\mathsf{ek}, \mathsf{Enc}(\mathsf{pk}, \mathsf{pf}, x)) = (g^{r'b}, g^{r'ab} \cdot H_{\mathbb{G}}(x))$. □

Theorem 3. *Def. 24 is a secure and correct* EO *scheme. More precisely, it is correct, satisfies pseudorandomness of the evaluation and ciphertext well-formedness, evaluated ciphertext simulatability, is randomized ciphertext indistinguishable as well as ciphertext indistinguishable for the evaluation and secret key owner. The latter two are semi-honest secure under the DDH assumption.*

Proof. Follows from Lemma 1, 2, 3, 4, 5, 6, and 7. □

C Three-Party Secure Shuffling for $\mathsf{D}_s\mathsf{PMC}$

C.1 Ideal Shuffle Functionality

The ideal shuffle functionality from Fig. 12 gets inputs from parties C and D secret shares and generates fresh shuffled shares and sends them back to parties S and D. Additionally, $\mathcal{F}_{\mathsf{Shuffle}}$ gets multiple EO ciphertexts from C, generates fresh shuffled ciphertexts, and sends them back to C. Parties P_1 to P_T do not participate in the protocol but do have information about the encrypted and secret shared information and might be corrupted.

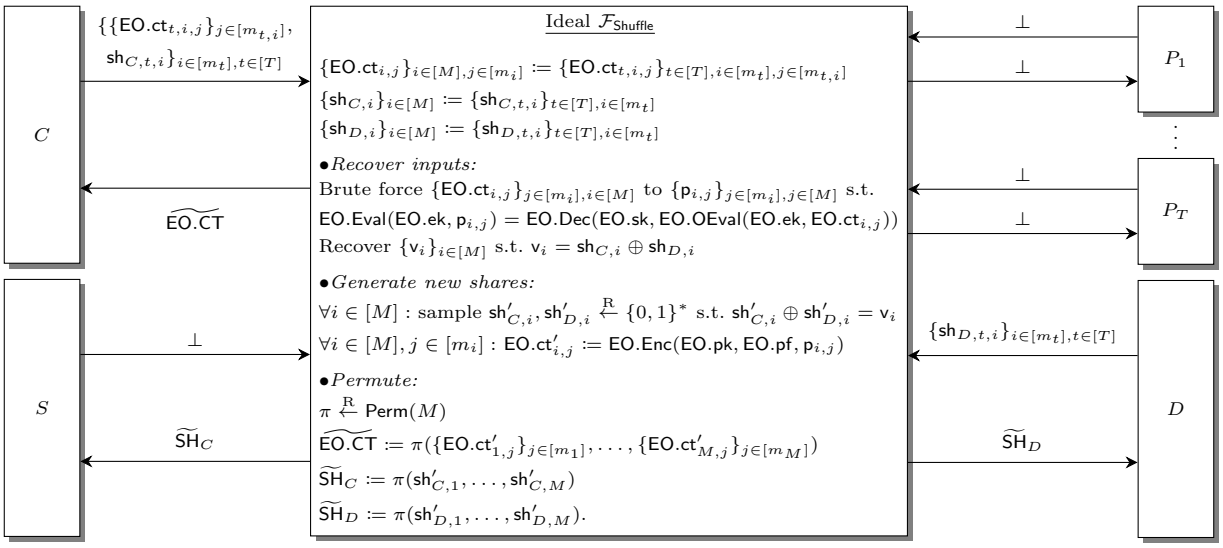


Fig. 12. The figure shows the ideal $\mathcal{F}_{\mathsf{Shuffle}}$ functionality. We define $M \coloneqq \sum_{t=1}^{T} m_t$. We treat $\mathsf{EO.pk}$ and $\mathsf{EO.pf}$ as publicly known to all parties. Party C has access to $\mathsf{EO.ek}$ and Party D to $\mathsf{EO.sk}$. Further, any amount of Parties P_1 to P_T can be corrupted who have access to $\{\mathsf{EO.ct}_{t,i,j}\}_{j\in[m_{t,i}]}$, $\{\mathsf{sh}_{C,t,i}\}_{i\in[m_t],t\in[T]}$ and $\{\mathsf{sh}_{D,t,i}\}_{i\in[m_t],t\in[T]}$.

C.2 Shuffle Protocol

We define a permutation of size m_C as an injective function $\pi : [N] \to [N]$. We denote as π_{AB} a permutation generated from party A and sent to B. Fig. 13 demonstrates the honest majority shuffling protocol utilized by $\mathsf{D}_s\mathsf{PMC}$. Our shuffling protocol performs two iterations of a permutation network and reshares C's and D's inputs (sh_C and sh_D, respectively). Parties C and D have T sh_C and sh_D vectors (indicated as $\mathsf{sh}_{C,t}, \mathsf{sh}_{D,t}$ for $t \in [T]$), each of which has m_t elements. Additionally, the shuffling protocol reshares $\mathsf{EO.ct}$ to prevent leakage of honest parties' data in the presence of an adversary that has corrupted D and multiple parties P.

The first iteration of the permutation network is demonstrated in steps 1-3 in Fig. 13 and reshares $\mathsf{sh}_C, \mathsf{sh}_D$ to $\overline{\mathsf{SH}}_C, \mathsf{SH}_S$ and $\mathsf{EO.ct}$ to $\overline{\mathsf{EO.CT}}$. Party C generates two permutations (π_{CS} and π_{CD}) as well as two vectors of scalars (V_{CS} and V_{CD}) to rerandomize sh_C and sh_D. C locally applies the two permutations and $\mathtt{XOR}$s with the vectors of scalars. C then sends one permutation and one vector of scalars to each of D and S. D first permutes and $\mathtt{XOR}$s sh_D with V_{CD} and sends the result to S who, in turn, permutes it with πCS and $\mathtt{XOR}$s it with V_{CS} to compute SH_S.

In the second iteration, party S generates two more permutations (π_{SC} and π_{SD}) as well as two vectors of scalars (V_{SC} and V_{SD}) to rerandomize the outputs of the first iteration (i.e., $\overline{\mathsf{SH}}_C$ and SH_S). Next, S applies both permutations on SH_S and $\mathtt{XOR}$s it with both vectors V_{SC} and V_{SD}, while parties C and D communicate to apply the same operations on $\overline{\mathsf{SH}}_C$. At the end of the protocol, S gets $\widetilde{\mathsf{SH}}_C$ and D gets $\widetilde{\mathsf{SH}}_D$ such that $\widetilde{\mathsf{SH}}_C \oplus \widetilde{\mathsf{SH}}_D = \mathsf{sh}_C \oplus \mathsf{sh}_D$. Finally, party C gets $\widetilde{\mathsf{EO.CT}}$, which is the blinded and rerandomized $\mathsf{EO.ct}$.

Observe that the communication in the aforementioned protocol is only linear to the size of $\mathsf{EO.CT}$. We can further optimize the communication by having each two parties (C with D, C with S, and S with D) pre-share some randomness and use it as a PRF key. These PRF keys can then be used to generate both the random permutations and the random vectors of scalars which will be consistent between the parties.

C.3 Security Analysis of Secure Shuffling

Theorem 4. *Let* EO *be a correct Rerandomizable Encrypted OPRF scheme that satisfies statistical rerandomized ciphertext indistinguishability, ciphertext indistinguishability (for evaluation key or secret key owner) and ciphertext well-formedness. Then, the shuffling protocol in Fig. 13 realizes the ideal shuffling functionality*

We define $M := \sum_{t=1}^{T} m_t$.

① First Shuffling **(Party C)**

Input: $\{\{\mathsf{EO.ct}_{t,i,j}\}_{j\in[m_{t,i}]}, \mathsf{sh}_{C,t,i}\}_{i\in[m_t],t\in[T]}$

1: $V_{CD}, V_{CS} \xleftarrow{R} \{0,1\}^{M\cdot|v|}$
2: $\pi_{CD}, \pi_{CS} \xleftarrow{R} \mathsf{Perm}(M)$
3: **For** $t \in [T], i \in [m_t], j \in [m_{t,i}]$: ▷ Randomize
4: $\quad \mathsf{EO.ct}'_{t,i,j} := \mathsf{EO.Rnd}(\mathsf{EO.pk}, \mathsf{EO.pf}, \mathsf{EO.ct}_{t,i,j})$
5: $\mathsf{SH}_C := (\mathsf{sh}_{C,1,1}, \ldots, \mathsf{sh}_{C,m_T,T})$
6: $\mathsf{EO.CT} := (\{\mathsf{EO.ct}'_{1,1,j}\}_j, \ldots, \{\mathsf{EO.ct}'_{T,m_T,j}\}_j)$
7: $\overline{\mathsf{SH}}_C := \pi_{CS}(\pi_{CD}(\mathsf{SH}_C) \oplus V_{CD}) \oplus V_{CS}$ ▷ Perm. & Rand.
8: $\overline{\mathsf{EO.CT}} := \pi_{CS}(\pi_{CD}(\mathsf{EO.CT}))$ ▷ Permute

Send to S: $\pi_{CS}, V_{CS}, \overline{\mathsf{EO.CT}}$
Send to D: π_{CD}, V_{CD}
Output of first shuffle: $\overline{\mathsf{SH}}_C$

② First Shuffling **(Party D)**

Input: $\{\mathsf{sh}_{D,t,i}\}_{i\in[m_t],t\in[T]}$
Messages: V_{CD}, π_{CD}

1: $\mathsf{SH}_D := (\mathsf{sh}_{D,1,1}, \ldots, \mathsf{sh}_{D,T,m_T})$
2: $\overline{\mathsf{SH}}_D := \pi_{CD}(\mathsf{SH}_D) \oplus V_{CD}$ ▷ Permute and Randomize

Send to S: $\overline{\mathsf{SH}}_D$
Output of first shuffle: –

③ First Shuffling **(Party S)**

Input: –
Messages: $\pi_{CS}, V_{CS}, \overline{\mathsf{SH}}_D, \overline{\mathsf{EO.CT}}$

1: $\mathsf{SH}_S := \pi_{CS}(\overline{\mathsf{SH}}_D) \oplus V_{CS}$ ▷ Permute and Randomize

Output of first shuffle: $\mathsf{SH}_S, \overline{\mathsf{EO.CT}}$

④ Second Shuffling **(Party S)**

Input: $\mathsf{SH}_S, \overline{\mathsf{EO.CT}}$

1: $(\{\overline{\mathsf{EO.ct}}_{1,j}\}_j, \ldots, \{\overline{\mathsf{EO.ct}}_{M,j}\}_j) := \overline{\mathsf{EO.CT}}$
2: $V_{SC}, V_{SD} \xleftarrow{R} \{0,1\}^{M\cdot|v|}$
3: $\pi_{SC}, \pi_{SD} \xleftarrow{R} \mathsf{Perm}(M)$
4: **For** $i \in [M], j \in [m_i]$: ▷ Randomize
5: $\quad \overline{\mathsf{EO.ct}}'_{i,j} := \mathsf{EO.Rnd}(\mathsf{EO.pk}, \mathsf{EO.pf}, \overline{\mathsf{EO.ct}}_{i,j})$
6: $\overline{\mathsf{EO.CT}}' := (\{\overline{\mathsf{EO.ct}}_{1,j}\}_j, \ldots, \{\overline{\mathsf{EO.ct}}_{M,j}\}_j)$
7: $\widetilde{\mathsf{SH}}_C = \pi_{SD}(\pi_{SC}(\mathsf{SH}_S) \oplus V_{SC}) \oplus V_{SD}$ ▷ Perm. & Rand.
8: $\widetilde{\mathsf{EO.CT}} := \pi_{SD}(\pi_{SC}(\overline{\mathsf{EO.CT}}))$ ▷ Permute

Send to C: $\pi_{SC}, V_{SC}, \widetilde{\mathsf{EO.CT}}$
Send to D: π_{SD}, V_{SD}
Output: $\widetilde{\mathsf{SH}}_C$

⑤ Second Shuffling **(Party C)**

Input: $\overline{\mathsf{SH}}_C$
Messages: $\pi_{SC}, V_{SC}, \widetilde{\mathsf{EO.CT}}$

1: $\widehat{\mathsf{SH}}_C := \pi_{SC}(\overline{\mathsf{SH}}_C) \oplus V_{SC}$ ▷ Permute and Randomize.

Send to D: $\widehat{\mathsf{SH}}_C$
Output: $\widetilde{\mathsf{EO.CT}}$

⑥ Second Shuffling **(Party D)**

Input: –
Messages: $\pi_{SD}, V_{SD}, \widehat{\mathsf{SH}}_C$

1: $\widetilde{\mathsf{SH}}_D := \pi_{SD}(\widehat{\mathsf{SH}}_C) \oplus V_{SD}$ ▷ Permute and Randomize

Output: $\widetilde{\mathsf{SH}}_D$

Fig. 13. Three-Party Shuffling. Parties C and D get secret shares sh_C and sh_D of a vector v as inputs such that $v = \mathsf{sh}_C \oplus \mathsf{sh}_D$. Party C additionally inputs a Rerandomizable Encrypted OPRF ciphertext vector $\mathsf{EO.ct}$ of same length as sh_C and sh_D. The protocol reshares $(\mathsf{sh}_C, \mathsf{sh}_D)$ to $(\widetilde{\mathsf{SH}}_C, \widetilde{\mathsf{SH}}_D)$ and carries along $\mathsf{EO.ct}$ and reshares it to $\widetilde{\mathsf{EO.CT}}$.

in Fig. 12 when at most one of the parties C, D and S and any amount of the parties P_1 to P_t are corrupted and semi-honest.

Proof. We prove the theorem by showing that for each party, there exists a simulator that produces a view that is indistinguishable from the view of the corrupted party in the real shuffle protocol.

Claim. Let EO be correct, satisfy statistical randomized ciphertext indistinguishability and ciphertext well-formedness. Then, there is a simulator that produces a view of Party C that is indistinguishable from the real view of Party C for any amount of corrupted parties P_1 to P_T. We emphasize that the distinguisher also receives the in and outputs to and from the ideal functionality (which is identical to the real output) of the honest parties.

Proof. We first show the simulator in case none of the parties P_1 to P_T is corrupted. The view of Party C can be generated from its input $\{\{\mathsf{EO.ct}_{t,i,j}\}_{j\in[m_{t,i}]}, \mathsf{sh}_{C,t,i}\}_{i\in[m_t],t\in[T]}$, output $\widetilde{\mathsf{EO.CT}}$ and the message $\pi_{SC}, V_{SC}, \widetilde{\mathsf{EO.CT}}$ from Party S. Our simulator emulates these messages and otherwise follows the description of the computation of Party C.

Our simulator receives input $\{\{\mathsf{EO.ct}_{t,i,j}\}_{j\in[m_{t,i}]}, \mathsf{sh}_{C,t,i}\}_{i\in[m_t],t\in[T]}$, $\widetilde{\mathsf{EO.CT}}$ and generates Party S's message as follows. It uses $\widetilde{\mathsf{EO.CT}}$ that was part of the input and samples $\pi_{SC} \xleftarrow{R} \mathsf{Perm}(M)$ and $V_{SC} \xleftarrow{R} \{0,1\}^{M\cdot|v|}$.

We now show that this simulator emulates the correct distribution. Let

$$\widetilde{\mathsf{EO.CT}} = \pi(\{\{\mathsf{EO.ct}_{t,i,j}\}_{j\in[m_{t,i}]}\}_{i\in[m_t],t\in[T]}) = \pi'_{SD}(\pi'_{SC}(\pi'_{CS}(\pi'_{CD}(\{\{\mathsf{EO.ct}_{t,i,j}\}_{j\in[m_{t,i}]}\}_{i\in[m_t],t\in[T]})))),$$

where π'_{SD}, $\pi'_{SC}, \pi'_{CS}, \pi'_{CD}$ are defined as in the original protocol and π'_{SC}, π'_{CS}, π'_{CD} are part of party C's view. Sampling $\pi'_{SD}, \pi'_{SC}, \pi'_{CS}, \pi'_{CD} \xleftarrow{R} \mathsf{Perm}(M)$ and defining π as their composition results in the same distribution as when sampling $\pi, \pi'_{SC}, \pi'_{CS}, \pi'_{CD} \xleftarrow{R} \mathsf{Perm}(M)$ and defining π'_{SD} such that it is consistent with the protocol specification. The former is the distribution during a real protocol execution while the later is the distribution during the simulated run where the ideal functionality samples π and the simulator samples $\pi'_{SC}, \pi'_{CS}, \pi'_{CD}$. π and π'_{SD} remain hidden from the view of Party C.

We follow this argument for the distribution of V_{SC}. There exists a unique $V \in \{0,1\}^{M\cdot|v|}$ such that $\mathsf{SH}_D = \mathsf{SH}'_D \oplus V$, where SH_D denotes the original shares sent by Party D to the ideal functionality and SH'_D are the shares generated and output by the ideal functionality. The same holds for SH_C and SH'_C. Further, as specified by the protocol V can also be defined as $V \coloneqq V_{SD} \oplus V_{SC} \oplus V_{CS} \oplus V_{CD}$. Here we ignore the fact that V is actually impacted by the permutations $\pi_{SD}, \pi_{SC}, \pi_{CS}, \pi_{CD}$ since it can simply be accounted for by permuting $V_{SD}, V_{SC}, V_{CS}, V_{CD}$. Both definitions of V are consistent since any two two out of two secret shares result in the same shares up to an offset vector in $\{0,1\}^{M\cdot|v|}$. As previously sampling first $V_{SD}, V_{SC}, V_{CS}, V_{CD}$ results in the same distribution as sampling first $V, V_{SC}, V_{CS}, V_{CD}$.

The last part to show is that the output $\widetilde{\mathsf{EO.CT}}$ of the ideal functionality is identically distributed as the Party C's output in the real execution. From the statistical randomized ciphertext indistinguishability of EO follows that any rerandomized ciphertext for input $\mathsf{p}_{i,j}$ is indistinguishable from a fresh encryption of $\mathsf{p}_{i,j}$ even when given $\mathsf{EO.ek}$ (and $\mathsf{EO.sk}$). Using a hybrid argument over all $N = \sum_{i=1}^{M}(m_i)$ (i.e., M total rows and each row i has m_i identifiers) distinguishing the real from the simulated view with advantage ϵ results in a ϵ/N distinguishing advantage in the randomized ciphertext indistinguishability game. Now, we show that brute forcing a $\mathsf{p}'_{i,j}$ from a ciphertext and encrypting it is except negligible probability identically distributed as a ciphertext of $\mathsf{p}_{i,j}$. By the correctness property it follows that except negligible probability, both ciphertexts evaluate to the same OPRF evaluation, i.e., $\mathsf{EO.Eval}(\mathsf{EO.ek}, \mathsf{p}_{i,j}) = \mathsf{EO.Eval}(\mathsf{EO.ek}, \mathsf{p}'_{i,j})$. Now, we can invoke the ciphertext well-formedness which ensures that the rerandomized $\widetilde{\mathsf{EO.CT}}$ is with overwhelming probability identically distributed as the fresh $\widetilde{\mathsf{EO.CT}}$ generated by the ideal functionality.

In case some of the parties P_1 to P_T are corrupted we actually do not need to adapt our simulator. The difference is that when adding the views of the corrupted parties among P_1 to P_T to the view of C, Party C has access to some of the shares $\{\mathsf{SH}_{D,t,i}\}_{i\in[m_t],t\in[T]}$. However, knowing these shares do not have impact on the distribution of the view generated by our simulator and can therefore simply added to the view. □

Claim. There exists a simulator that produces a view of Party D that is indistinguishable from the real view of Party D for any amount of corrupted parties P_1 to P_T.

Proof. We start with the case where there is no corruption among parties P_1 to P_T. Party D's view can be generated from its input $\{\mathsf{sh}_{D,t,i}\}_{i\in[m_t]t\in[T]}$, output $\widetilde{\mathsf{SH}}_D$ and the messages (π_{CD}, V_{CD}), $\widehat{\mathsf{SH}}_C$ from Party C and π_{SD}, V_{SD} from Party S. Therefore it suffices for our simulator to emulate these messages and generate the view from these messages according to the protocol description.

Our simulator on input $\{\mathsf{sh}_{D,t,i}\}_{i\in[m_t]t\in[T]}$, $\widetilde{\mathsf{SH}}_D$ samples π_{CD}, $\pi_{SD} \xleftarrow{R} \mathsf{Perm}(M)$, $V_{CD}, V_{SD}, \xleftarrow{R} \{0,1\}^{M\cdot|v|}$. $\widehat{\mathsf{SH}}_C$ is picked such that $\widetilde{\mathsf{SH}}_D = \pi_{SD}(\widehat{\mathsf{SH}}_C) \oplus V_{SD}$. We define π as in the previous claim. As previously, sampling first $\pi_{SD}, \pi_{SC}, \pi_{CS}, \pi_{CD}$ and defining π as their composition as done in the real protocol execution results in the same distribution as when sampling π, π_{SD}, π_{CD} first and then defining and sampling π_{SC}, π_{CS} (not part of the view) such that they are consistent with the real protocol distribution. Using the same approach, we can show that V_{CD}, V_{SD} are also correctly distributed.

Similar to the previous claim, corrupting any amount of parties P_1 to P_T and adding them to the view of Party D does not impact the distribution of the view generated by the simulator. Again, we can simply add $\{\{\mathsf{EO.ct}_{t,i,j}\}_{j\in[m_{t,i}]}, \mathsf{sh}_{C,t,i}\}_{i\in[m_t],t\in[T]}$ of the corrupted parties to the view generated by our simulator. □

Claim. Let EO satisfy statistical rerandomized ciphertext indistinguishability and ciphertext indistinguishability (for evaluation key or secret key owner). Then, there is a simulator that produces a view of Party S that is indistinguishable from the real view of Party S for any amount of corrupted parties P_1 to P_T.

Proof. Let $\{\{\mathsf{EO.ct}'_{t,i,j}\}_{j\in[m_{t,i}]}, \mathsf{sh}'_{C,t,i}, \mathsf{sh}'_{D,t,i}\}_{i\in[m_t],t\in\mathbb{C}\subseteq[T]}$ be the views of the corrupted parties among P_1 to P_T. The view of Party S and the corrupted parties among P_1 and P_T can be generated from $\{\{\mathsf{EO.ct}'_{t,i,j}\}_{j\in[m_{t,i}]}, \mathsf{sh}'_{C,t,i}, \mathsf{sh}'_{D,t,i}\}_{i\in[m_t],t\in\mathbb{C}\subseteq[T]}$, S's output $\widetilde{\mathsf{SH}}_C$, the messages $\pi_{CS}, V_{CS}, \overline{\mathsf{EO.CT}}$ from Party D and $\overline{\mathsf{SH}}_D$ from Party D.

Our simulator has inputs $\{\{\mathsf{EO.ct}'_{t,i,j}\}_{j\in[m_{t,i}]}, \mathsf{sh}'_{C,t,i}, \mathsf{sh}'_{D,t,i}\}_{i\in[m_t],t\in\mathbb{C}\subseteq[T]}$ and $\widetilde{\mathsf{SH}}_C$. It samples $\pi_{CS} \xleftarrow{\text{R}} \mathsf{Perm}(M)$, $V_{CS} \xleftarrow{\text{R}} \{0,1\}^{M\cdot|v|}$ and generates $\overline{\mathsf{EO.CT}}$ as encryptions of 0.

We now show that the simulator generates the correct distribution. We define π as in the previous claims. $\pi_{SD}, \pi_{SC}, \pi_{CS}, \pi_{CD}$ are part of the simulated view except π and π_{CD}. By using the same sampling argument as before, π_{CS} and V_{CS} follow the correct distribution.

It remains to show that $\overline{\mathsf{EO.CT}}$ are distributed correctly. We use a hybrid argument to show this.

Hybrid_0**:** The first hybrid defines $\overline{\mathsf{EO.CT}}$ according to the real execution. In the real execution, Party C uses the $\mathsf{EO.Rnd}$ procedure to rerandomize $\{\mathsf{EO.ct}'_{t,i,j}\}_{j\in[m_{t,i}],i\in[m_t],t\in[T]}$ and applies the permutations π_{CD} and π_{CS} the outcome is $\overline{\mathsf{EO.CT}}$.

Hybrid_1 This hybrid generates $\overline{\mathsf{EO.CT}}$ as a fresh encryption of $\mathsf{p}_{t,i,j}$ using $\mathsf{EO.Enc}(\mathsf{EO.pk}, \mathsf{EO.pf}, \mathsf{p}_{t,i,j})$.

Hybrid_2**:** The last hybrid generates $\overline{\mathsf{EO.CT}}$ as an encryption of 0 using $\mathsf{EO.Enc}(\mathsf{EO.pk}, \mathsf{EO.pf}, 0)$.

Based on the statistical ciphertext indistinguishability of EO, Hybrid_0 and Hybrid_1 generate up to negligible probability the same distribution. We can use a standard hybrid argument to show this. Let ϵ be the distinguishing probability between Hybrid_0 and Hybrid_1 and let $N = \sum_{i=1}^{M} m_i$ be the amount of ciphertexts, then the statistical ciphertext indistinguishability can be broken with probability $\frac{\epsilon}{N}$.

We show now that Hybrid_1 and Hybrid_2 are computationally indistinguishable based on the ciphertext indistinguishability (for secret key or evaluation key owner). The two notions give the adversary access to either $\mathsf{EO.sk}$ or $\mathsf{EO.ek}$. Since the corrupted parties among P_1 to P_T as well as Party S do not have access to either of the keys, a weaker notion suffices in which no access to $\mathsf{EO.sk}$, $\mathsf{EO.ek}$ is given. This weaker notion is implied by both of the ciphertext indistinguishability notions of an EO scheme.

We use a standard hybrid in which we replace step by step one of the N ciphertexts with an encryption with 0. The last hybrid matches Hybrid_2 and the first hybrid Hybrid_1. For each step we use a reduction to the ciphertext indistinguishability game in which given $\mathsf{EO.pk}$, $\mathsf{EO.pf}$, we need to construct a distinguisher D' that distinguishes between an encryption of $x_0 = \mathsf{p}_{t,i,j}$ and $x_1 = 0$. We construct this distinguisher by invoking the distinguisher D between two intermediate hybrids. D' forwards $\mathsf{EO.pk}$ and $\mathsf{EO.pf}$, it generates the view of the corrupted parties as specified by the hybrids with the exception of the one ciphertext that is different in the hybrids. D' uses the challenge ciphertext for this ciphertext. Finally D' outputs the output of D.

If D successfully distinguishes two intermediate hybrids, D' breaks the ciphertext indistinguishability for the secret key and evaluation key owner of the EO scheme. Let ϵ be an upper bound on the distinguishing probability in the ciphertext indistinguishability game. Then the distinguishing probability between Hybrid_1 and Hybrid_2 is upper bounded by ϵ/N.

The indistinguishability between Hybrid_0, Hybrid_1, and Hybrid_2 concludes our claim. □

□

D Security Analysis

D.1 Security Analysis of DPMC

Proof. We prove Theorem 1 by proving the following two claims.

Claim. Let the secret key encryption and the PKE scheme be IND-CPA secure, the KEM simulatable and the DDH assumption hold.

Then there exists a simulator that generates the joint view of Party C and any subset of parties P_1 to P_T that is computationally indistinguishable from the real view.

Proof. The joint view of Party C and the subset of corrupted parties among P_1 to P_T, identified by $\mathbb{C} \subseteq [T]$ can be generated from their inputs KV_C, $\mathsf{KV}_{t\in\mathbb{C}}$, the outputs SH_C, and the messages $\{\mathsf{cta}_t, \mathsf{ctb}_t, \{\{\mathsf{ha}_{t,i,j}\}_{j\in[m_{t,i}]}, \mathsf{ctc}_{t,i}\}_{i\in[m_t]}\}_{t\in[T]}$, $\{\widehat{\mathsf{KEM.cp}}_{i,t}\}_{i\in[m_C],t\in[T]}$.

The simulator on input KV_C, $\{\mathsf{KV}_t\}_{t\in\mathbb{C}}$, and SH_C simulates the messages as follows. It samples $(\mathsf{KEM.pk}, \mathsf{KEM.sk}) \leftarrow \mathsf{KEM.KG}(1^\kappa)$ and uses $\mathsf{KEM.Sim}$ on input $\mathsf{KEM.sk}, \mathsf{SH}_C$ to compute message $\{\widehat{\mathsf{KEM.cp}}_{i,t}\}_{i\in[m_C],t\in[T]}$. For all $t \notin \mathbb{C}$, it samples $\mathsf{sk}_t \leftarrow \mathsf{SKE.KG}(1^\kappa)$, $\mathsf{cta}_t \leftarrow \mathsf{PKE.Enc}(\mathsf{pk}_D, 0)$, $\mathsf{ctb}_t \leftarrow \mathsf{SKE.Enc}(\mathsf{sk}_t, 0)$, $\mathsf{ctc}_{t,i} \leftarrow \mathsf{SKE.Enc}(\mathsf{sk}_t, 0)$, $r_{t,i,j} \xleftarrow{R} \mathbb{Z}_q$ and defines $\mathsf{ha}_{t,i,j} \coloneqq g^{r_{t,i,j}}$.

We use the following sequence of hybrids to show that the joint view during the real execution is indistinguishable from the view generated by the simulator.

Hybrid_1**:** Identical to the view during the real protocol execution.
Hybrid_2**:** Computes $(\mathsf{KEM.pk}, \mathsf{KEM.sk}) \leftarrow \mathsf{KEM.KG}(1^\kappa)$ as output of $\mathsf{KEM.Sim}$ on input $\mathsf{KEM.sk}$, SH_C.
Hybrid_3**:** For all $t \in \mathbb{C}$, compute cta_t as $\mathsf{cta}_t \leftarrow \mathsf{PKE.Enc}(\mathsf{pk}_D, 0)$.
Hybrid_4**:** For all $t \in \mathbb{C}$, compute ctb_t, $\mathsf{ctc}_{t,i}$ as $\mathsf{ctb}_t \leftarrow \mathsf{SKE.Enc}(\mathsf{sk}_t, 0)$, $\mathsf{ctc}_{t,i} \leftarrow \mathsf{SKE.Enc}(\mathsf{sk}_t, 0)$.
Hybrid_5**:** For all $t \in \mathbb{C}$, compute $\mathsf{ha}_{t,i,j}$ as $\mathsf{ha}_{t,i,j} \coloneqq g^{r_{t,i,j}}$ where $r_{t,i,j} \xleftarrow{R} \mathbb{Z}_q$.

Hybrid_1 and Hybrid_2 are indistinguishable except with negligible probability based on the simulatability of the key encapsulation scheme.

Hybrid_2 and Hybrid_3 are indistinguishable based on the IND-CPA security of the PKE scheme. Notice that only party D has access to sk_D. The reduction works as follows. Let there be a distinguisher against Hybrid_2 and Hybrid_3 with probability ϵ. Then, we define a sequence of $T+1$ hybrids in which we step by step replace cta_t with encryptions of 0. The distinguisher can distinguish at least one of the hybrids with at least probability ϵ/T. We can use it to construct a distinguisher against the IND-CPA game as follows. The distinguisher receives pk from the IND-CPA game and defines $\mathsf{pk}_D \coloneqq \mathsf{pk}$. It sets $x_0 \coloneqq \mathsf{sk}_t$ and $x_1 \coloneqq 0$ and receives back a challenge ciphertext ct. It defines $\mathsf{cta}_t \coloneqq \mathsf{ct}$. It outputs whatever the distinguisher between Hybrid_2 and Hybrid_3 outputs. This distinguisher breaks the IND-CPA security with probability ϵ/T. By the security of the PKE scheme, this must be negligible and therefore Hybrid_2 and Hybrid_3 can be distinguished with at most negligible probability as well.

Since cta is independent of the symmetric key, we can now use the IND-CPA security of the symmetric key encryption to replace ctb and ctc with encryptions of 0. Again, we define a sequence of hybrids in which we replace step by step the ciphertexts by encryptions of 0. The distinguisher against Hybrid_3 and Hybrid_4 can distinguish at least two consecutive intermediate hybrids with at least probability $\epsilon/(T + \sum_{t\in\mathbb{C}} m_t)$. The distinguisher against the IND-CPA game can use $x_0 \coloneqq a_t$ ($x_0 \coloneqq \mathsf{sh}_{D,t,i}$) and x_1 in the IND-CPA game for the challenge ciphertext. Then, the distinguisher can use the challenge ciphertext to either simulate the first or second consecutive intermediate hybrid and output whatever the distinguisher against the hybrids outputs. Therefore, Hybrid_3 and Hybrid_4 can be distinguished at most with negligible probability.

Notice that the ciphertexts are now independent of scalar a_t. We can use the DDH assumption (Def. 12) to argue that Hybrid_4 and Hybrid_5 are indistinguishable. Again, we use a sequence of hybrids in which we replace step by step $\mathsf{ha}_{t,i,j}$ with a uniform group element, i.e., $\mathsf{ha}_{t,i,j} \coloneqq g^{r_{t,i,j}}$ where $r_{t,i,j} \xleftarrow{R} \mathbb{Z}_q$. There are $\sum_{t\in\mathbb{C},i\in[m_t]} m_{t,i}$ hybrids. Let there be a distinguisher between Hybrid_4 and Hybrid_5 with probability ϵ. Then, there are two consecutive intermediate hybrids that this distinguisher distinguishes with at least probability $\epsilon/(\sum_{t\in\mathbb{C},i\in[m_t]} m_{t,i})$. The reduction to DDH works as follows. The DDH distinguisher receives challenge A, B, C. Before invoking the hybrid distinguisher, it programs $H_{\mathbb{G}}(\mathsf{p}_{t,i,j}) \coloneqq B$ and defines $\mathsf{h}_{t,i,j} \coloneqq C$. For all other $\mathsf{h}_{t,i,j}$ that are not uniform yet, it programs $H_{\mathbb{G}}(\mathsf{p}_{t,i,j}) \coloneqq g^{x_{t,i,j}}$, where $x_{t,i,j} \xleftarrow{R} \mathbb{Z}_q$ and defines $\mathsf{h}_{t,i,j} \coloneqq A^{x_{t,i,j}}$. The DDH distinguisher outputs the output of the hybrid distinguisher. When $A = g^a, B = g^b, C = g^{ab}$, all $\mathsf{h}_{t,i,j}$ are correctly defined as in the first consecutive hybrid. When A, B, C are uniform group elements, $\mathsf{h}_{t,i,j} = C$ is uniform while all other $\mathsf{h}_{t,i,j}$ are distributed according to the second (and first) of the consecutive hybrids. Therefore, Hybrid_4 and Hybrid_5 can be distinguished with at most negligible probability which concludes the proof of our claim. □

Claim. Let the KEM scheme be key indistinguishable and the DDH assumption hold.

Then there exists a simulator with access to the leakage defined in Def. 9 that generates the joint view of Party C and any subset of parties P_1 to P_T that is computationally indistinguishable from the real view.

Proof. The joint view of Party D and the subset of corrupted parties among P_1 to P_T, i.e., defined by $\mathbb{C} \subseteq [T]$ can be generated by the inputs sk_D, $\{\mathsf{KV}_t\}_{t\in\mathbb{C}}$, output SH_D and messages $\mathsf{KEM.pk}$, $\{(\mathsf{h}_{C,i,j})_{j\in[m_{C,i}]}\}_{i\in[m_C]}$ and $\{\mathsf{cta}_t, \mathsf{ctb}_t, \{\{\mathsf{hca}_{t,i,j}\}_{j\in[m_{t,i}]}, \mathsf{ctc}_{t,i}\}_{i\in[m_t]}\}_{t\in[T]}$.

Given the leakage defined in Def. 9 and inputs sk_D, $\{\mathsf{KV}_t\}_{t\in\mathbb{C}}$, SH_D, the simulator works as follows. The simulator uses the leakage to define $\{(\mathsf{h}_{C,i,j})_{j\in[m_{C,i}]}\}_{i\in[m_C]}$ and $\{\mathsf{hc}_{t,i,j}\}_{j\in[m_{t,i}],i\in[m_t],t\in[T]}$. For all $t\notin\mathbb{C}$, it samples $a_t \xleftarrow{R} \mathbb{Z}_q$ (for all other t, a_t is already defined when generating the view for P_t). $\mathsf{hca}_{t,i,j} := \mathsf{hc}_{t,i,j}^{a_t}$. For all $t\notin\mathbb{C}$, sample $\mathsf{sk}_t \leftarrow \mathsf{SKE.KG}(1^\kappa)$ and use sk_t and pk_D to define cta, ctb and ctc according to the protocol description, where $\mathsf{sh}_{D,t,i}$ is defined s.t. it is consistent with SH_D and $(\mathsf{KEM.cp}_{t,i}, \mathsf{KEM.k}_{t,i}) \leftarrow \mathsf{KEM.Enc}(\mathsf{KEM.pk})$.

We prove that the view generated by the simulator is indistinguishable from the real view using the following hybrids.

Hybrid_1**:** Is identical to the view during the protocol.
Hybrid_2**:** For all $t\notin\mathbb{C}$, generate $(\mathsf{KEM.cp}_{t,i}, \mathsf{KEM.k}_{t,i}) \leftarrow \mathsf{KEM.Enc}(\mathsf{KEM.pk})$. (Now $\mathsf{KEM.k}_{t,i}$ is independent of sh_C).
Hybrid_3**:** Use the leakage to define $\{(\mathsf{h}_{C,i,j})_{j\in[m_{C,i}]}\}_{i\in[m_C]}$ and $\{\mathsf{hc}_{t,i,j}\}_{j\in[m_{t,i}],i\in[m_t],t\in[T]}$.

Hybrid_1 and Hybrid_2 are indistinguishable based on the key indistinguishability of the key encapsulation. To show this, we use a sequence of hybrids in which we replace $\mathsf{KEM.cp}_{t,i}$ generated by P_t for $t\notin\mathbb{C}$ and related to $\mathsf{sh}_{C,t,i}$ with $(\mathsf{KEM.cp}'_{t,i}, \mathsf{KEM.k}'_{t,i}) \leftarrow \mathsf{KEM.Enc}(\mathsf{KEM.pk})$. We use the triangular inequality which implies that if $\mathsf{KEM.cp}, \mathsf{KEM.k}$ cannot be distinguished with more than probability ϵ from $\mathsf{KEM.cp}, u$ for a uniform u, $\mathsf{KEM.cp}, \mathsf{KEM.k}$ cannot be distinguished from $\mathsf{KEM.cp}', \mathsf{KEM.k}$ with more than probability 2ϵ. Let there be a distinguisher that distinguishes Hybrid_1 and Hybrid_2 with probability ϵ. Then it distinguishes at least two consecutive intermediate hybrids with probability $\epsilon/(\sum_{t\in\mathbb{C}} m_t)$. Given this distinguisher, we build a distinguisher against the key indistinguishability which receives challenge $\mathsf{KEM.cp}, \mathsf{KEM.k}$ and sets $\mathsf{sh}_{C,t,i} := \mathsf{KEM.k}$. The distinguisher outputs the output of the hybrid distinguisher. When $\mathsf{KEM.k}$ is consistent with $\mathsf{KEM.cp}$, the distinguisher simulates Hybrid_1 and otherwise Hybrid_2. This distinguisher breaks the key indistinguishability with probability $\epsilon/(2\sum_{t\in\mathbb{C}} m_t)$. Since this is negligible, Hybrid_1 and Hybrid_2 cannot be distinguished except negligible probability.

Hybrid_2 and Hybrid_3 are indistinguishable based on the DDH assumption. We show this by using a sequence of intermediate hybrids in which we replace $\{(\mathsf{h}_{C,i,j})_{j\in[m_{C,i}]}\}_{i\in[m_C]}$ and $\{\mathsf{hc}_{t,i,j}\}_{j\in[m_{t,i}],i\in[m_t],t\in[T]}$ with uniform group elements. If there is a distinguisher that distinguishes Hybrid_2 and Hybrid_3 with probability ϵ, then it distinguishes at least two consecutive intermediate hybrids with probability $\epsilon/(\sum_{i\in[m_C]} m_{C,i} + \sum_{t\in[T],i\in[m_t]} m_{t,i})$. The distinguisher against DDH receives A, B, C and defines $\mathsf{hca}_{t,i,j}^{1/a_t} := C$ ($\mathsf{h}_{C,i,j} := C$), programs $H_{\mathbb{G}}(\mathsf{p}_{t,i,j}) := B$ ($H_{\mathbb{G}}(c_{i,j}) := B$). For all $\mathsf{hca}_{t,i,j}$, $\mathsf{h}_{C,i,j}$ that are not uniform yet, program $H_{\mathbb{G}}(\mathsf{p}_{t,i,j}) := g^{x_{t,i,j}}$, $H_{\mathbb{G}}(\mathsf{c}_{i,j}) := g^{x_{i,j}}$ and define $\mathsf{hca}_{t,i,j}^{1/a_t} := A^{x_{t,i,j}}$, $\mathsf{h}_{C,i,j} := A^{x_{i,j}}$. When $A = g^a, B = g^b, C = g^{ab}$, the DDH distinguisher simulates the first of the intermediate hybrids otherwise the second one. Notice that in the latter case, $\mathsf{hc}_{t,i,j} := \mathsf{hca}_{t,i,j}^{1/a_t}$ ($\mathsf{h}_{C,i,j}$) is uniform. This concludes the proof of our claim. □

□

D.2 Security Analysis of $\mathsf{D}_s\mathsf{PMC}$

Proof. We prove Theorem 2 by constructing a simulator that can generate a view of the corrupted parties from their inputs and outputs that is indistinguishable from their view during a real execution. We emphasize that the distinguisher has access to the inputs and outputs of the honest parties specified by the ideal functionality in Fig. 2, which matches the outputs of the real protocol. We show this in the following three claims.

Claim. Let PKE be an IND-CPA secure and correct PKE scheme, PRG a secure pseudorandom generator and EO be a correct and satisfy statistical rerandomized ciphertext indistinguishability, the (semi-honest) ciphertext indistinguishability for the evaluation key owner and ciphertext well-formedness.

Then, there exists a simulator that generates the joint view of Party C and any subset of parties P_1 to P_T that is indistinguishable from the joint view during the protocol execution.

Proof. The joint view can be generated from the input and messages received by party C and the subset of parties P_1 to P_T. Let this subset be $\mathbb{C}\subseteq[T]$. Notice that the parties do not have any outputs as specified in the ideal functionality in Fig. 2.

The inputs are KV_C and $\{\mathsf{KV}_t\}_{t\in\mathbb{C}}$ and the output is SH_C. The parties P_1 to P_T receive messages $\mathsf{EO.pk}, \mathsf{EO.pf}$ and have access to pk_D, where $\mathsf{EO.pf}$ is generated by Party C. Party C receives the messages $\{\{\{\mathsf{EO.ct}_{t,i,j}\}_{j\in[m_{t,i}]}, \mathsf{sh}_{C,t,i}\}_{i\in[m_t]}, \mathsf{cta}_t\}_{t\in[T]}$ and $\{\widehat{\mathsf{KEM.cp}}_{i,t}\}_{i\in[m_c],t\in[T]}$.

The simulator receives input KV_C, $\{\mathsf{KV}_t\}_{t\in\mathbb{C}}$, SH_C and emulates the view as follows. It samples $(\mathsf{KEM.pk}, \mathsf{KEM.sk}) \leftarrow \mathsf{KEM.KG}(1^\kappa)$, $(\mathsf{EO.pk}, \mathsf{EO.sk}) \leftarrow \mathsf{EO.KG}(1^\kappa)$ and $(\mathsf{pk}_D, \mathsf{sk}_D) \leftarrow \mathsf{PKE.KG}(1^\kappa)$. It samples $\mathsf{cta}_t \leftarrow \mathsf{PKE.Enc}(\mathsf{pk}, 0)$ for all $t \notin [T]$. It samples $\mathsf{sh}_{C,t,i} \xleftarrow{R} \{0,1\}^{|\mathsf{v}|}$ for all $t \notin \mathbb{C}$. It defines $\widehat{\mathsf{sh}}_{C,i,t}$ consistently with SH_C for all $t \in [T]$ and defines $\widehat{\mathsf{KEM.cp}}_{i,t} \leftarrow \mathsf{KEM.Sim}(\mathsf{KEM.sk}, \widehat{\mathsf{sh}}_{C,i,t})$. Further, it defines $\mathsf{EO.ct}_{t,i,j} \leftarrow \mathsf{EO.Enc}(\mathsf{EO.pk}, \mathsf{EO.pf}, 0)$ for all $t \notin \mathbb{C}$. For $t \in \mathbb{C}$, $\mathsf{EO.ct}_{t,i,j} \leftarrow \mathsf{EO.Enc}(\mathsf{EO.pk}, \mathsf{EO.pf}, \mathsf{p}_{t,i,j})$, where $\mathsf{p}_{t,i,j} \in \mathsf{KV}_t$.

We use the following sequence of hybrids to show that the simulated view is indistinguishable from the view during the real protocol execution.

Hybrid_0**:** Identical to the view during the real protocol execution.

Hybrid_1**:** Samples $\mathsf{cta}_t \xleftarrow{\mathsf{PKE.Enc}} (\mathsf{pk}, 0)$ for all $t \notin \mathbb{C}$.

Hybrid_2**:** Samples $\mathsf{sh}_{D,t,i} \xleftarrow{R} \{0,1\}^{|\mathsf{v}|}$ for all $t \notin \mathbb{C}$ (instead of using PRG).

Hybrid_3**:** Invoke the simulator of the shuffling protocol to simulate the view during the shuffling. The input $\{\{\mathsf{EO.ct}_{t,i,j}\}_{j\in[m_{t,i}]}, \mathsf{sh}_{C,t,i}\}_{i\in[m_t]}$, $\{\widetilde{\mathsf{EO.ct}}_{i,j}\}_{i\in[M],j\in[m_i]}$ of the simulator is distributed as in Hybrid_2. Notice that the simulator also receives $\mathsf{EO.pk}$, $\mathsf{EO.pf}$ and $\mathsf{EO.ek}$.

Hybrid_4**:** Replaces $\{\mathsf{EO.ct}_{t,i,j}\}_{j\in[m_{t,i}]}$ for all $t \notin \mathbb{C}$ and all $\{\widetilde{\mathsf{EO.ct}}_{i,j}\}_{i\in[M],j\in[m_i]}$ with independent encryptions of 0. More precisely, $\mathsf{EO.ct}_{t,i,j} \leftarrow \mathsf{EO.Enc}(\mathsf{EO.pk}, \mathsf{EO.pf}, 0)$ and $\widetilde{\mathsf{EO.ct}}_{i,j} \leftarrow \mathsf{EO.Enc}(\mathsf{EO.pk}, \mathsf{EO.pf}, 0)$.

Hybrid_5**:** Samples $\mathsf{sh}_{C,t,i} \xleftarrow{R} \{0,1\}^{|\mathsf{v}|}$ for all $t \notin \mathbb{C}$. Further, defines $\widehat{\mathsf{sh}}_{C,i,t}$ consistently with SH_C and samples $\widehat{\mathsf{KEM.cp}}_{i,t} \leftarrow \mathsf{KEM.Sim}(\mathsf{KEM.sk}, \widehat{\mathsf{sh}}_{C,i,t})$.

Notice that the view in Hybrid_5 is identically distributed as the view generated by the simulator.

We now show that the hybrids are indistinguishable. Let Hybrid_0 and Hybrid_1 be distinguishable with probability ϵ. We define a sequence of intermediate hybrids that replaces the ciphertexts $\mathsf{cta}_t \leftarrow \mathsf{PKE.Enc}(\mathsf{pk}, \mathsf{seed}_t)$ with $\mathsf{cta}_t \xleftarrow{\mathsf{PKE.Enc}} (\mathsf{pk}, 0)$. Then there is a distinguisher that distinguishes one of the intermediate hybrids with at least probability ϵ/T. Such a distinguisher would directly distinguish challenge ciphertexts for $x_0 := \mathsf{seed}_t$ from $x_1 := 0$ in the IND-CPA game of the PKE scheme. Therefore the distinguishing probability between Hybrid_0 and Hybrid_1 is upper bounded by the IND-CPA security of PKE.

Let Hybrid_1 and Hybrid_2 be distinguishable with probability ϵ. We define a sequence of intermediate hybrids in which we step by step replace $(\mathsf{sh}_{D,t,1}, \ldots, \mathsf{sh}_{D,t,m_t}) = \mathsf{PRG}(\mathsf{seed}_t)$ with $(\mathsf{sh}_{D,t,1}, \ldots, \mathsf{sh}_{D,t,m_t}) \leftarrow \{0,1\}^{m_t\cdot|\mathsf{v}|}$. Then, there would be a distinguisher that distinguishes two consecutive intermediate hybrids with at least probability ϵ/T. This would imply a distinguisher that breaks the security of the PRG with the same probability. Since the PRG is indistinguishable except negligible probability, Hybrid_0 and Hybrid_1 cannot be distinguished except negligible probability.

Let Hybrid_2 and Hybrid_3 be distinguishable with probability ϵ. Then, this would allow to distinguish the simulated view during the shuffle protocol from the real view. However, as shown in Theorem 4 this probability is upper bounded the correctness, the statistical rerandomized ciphertext indistinguishability, the (semi-honest) ciphertext indistinguishability (for evaluation key or secret key owner) and ciphertext well-formedness of the EO scheme. Therefore Hybrid_2 and Hybrid_3 cannot be distinguished beyond the bound given in the proof of Theorem 4.

We use the (semi-honest) ciphertext indistinguishability for the evaluation key owner to argue that Hybrid_3 and Hybrid_4 are indistinguishable. Notice that in the ideal shuffle functionality (see Fig. 12), the ciphertext sets $\{\mathsf{EO.ct}_{t,i,j}\}_{j\in[m_{t,i}]}$ and $\{\widetilde{\mathsf{EO.ct}}_{i,j}\}_{i\in[M],j\in[m_i]}$ are independent encryptions. Therefore, we can replace them independently with encryptions of 0. We need to use the ciphertext indistinguishability for the evaluation key owner since the simulator need access to $\mathsf{EO.ek}$ which is also used by the simulator of the shuffling protocol. The indistinguishability between Hybrid_3 and Hybrid_4 follows from a straightforward reduction to the ciphertext indistinguishability using a hybrid argument in which we replace step by step each ciphertext with an encryption of 0 until all ciphertexts are encryptions of 0. If there exists a distinguisher between Hybrid_3 and Hybrid_4 that distinguishes them with probability ϵ, then there is a distinguisher that distinguishes one of the intermediate hybrids with at least probability $\epsilon/2N$, where N is the size of $\{\mathsf{KV}_t\}_{t\in[T]}$. The distinguisher for the intermediate hybrids would then lead to a distinguisher against the ciphertext indistinguishability for the evaluation key owner of the EO scheme.

We finalize the claim by showing the indistinguishability of Hybrid_4 and Hybrid_5. Similar as in case of the ciphertexts, the ideal shuffle functionality samples the shares $\mathsf{sh}_{C,t,i}$ and $\widehat{\mathsf{sh}}_{C,i,t}$ independently. Therefore, we can also sample them independently. Hybrid_5 generates statistically the same view as Hybrid_4 for the following reason. Sampling $\mathsf{sh}_C \xleftarrow{R} \{0,1\}^{|\mathsf{v}|}$ and $\mathsf{sh}_D \xleftarrow{R} \{0,1\}^{|\mathsf{v}|}$ under the constraint that $\mathsf{sh}_C \oplus \mathsf{sh}_D = \mathsf{v}$ (Hybrid_4) results in the same distribution as when sampling $\mathsf{sh}_C \xleftarrow{R} \{0,1\}^{|\mathsf{v}|}$ and defining $\mathsf{sh}_D := \mathsf{v} \oplus \mathsf{sh}_C$ (Hybrid_5), where sh_D and v are not known to the simulator. Thus, sh_C can be sampled independently of sh_D and v by sampling $\mathsf{sh}_C \xleftarrow{R} \{0,1\}^{|\mathsf{v}|}$. Further, by the property of $\mathsf{KEM.Sim}$, $\widehat{\mathsf{KEM.cp}}_{i,t}$ has the same distribution when being an output of $\mathsf{KEM.Enc}$ and $\mathsf{KEM.Sim}$. This concludes our claim.

Claim. Let PKE be a correct PKE scheme, KEM a secure and correct key encapsulation scheme and EO secure, correct and evaluated ciphertext simulatable.

Then, there exists a simulator with access to the leakage graph of Def. 11 that generates the joint view of Party D and any subset of parties P_1 to P_T that is indistinguishable from the joint view during the protocol execution.

Proof. The joint view of Party D and the subset of parties P_1 to P_T (defined by $\mathbb{C} \subseteq [T]$) can be generated from inputs $(\mathsf{pk}_D, \mathsf{sk}_D)$, $\{\mathsf{KV}_t\}_{t\in\mathbb{C}}$, output SH_D and messages $\mathsf{KEM.pk}$, $\mathsf{EO.pf}$, $\{\mathsf{cta}_t\}_{t\notin\mathbb{C}}$ and $\{\mathsf{KEM.cp}_i, \overline{\mathsf{sh}}_{D,i}\}_{i\in[M]}$, $\{\mathsf{h}_{C,i,j}\}_{i\in[m_C],j\in[m_{C,i}]}$, $\{\mathsf{EO.ect}_{i,j}\}_{i\in[M],j\in[m_i]}$. Further, the view depends on the leakage graph defined in Def. 11.

The simulator emulates the joint views as follows. It samples $(\mathsf{KEM.pk}, \mathsf{KEM.sk}) \leftarrow \mathsf{KEM.KG}(1^\kappa)$ and $(\mathsf{EO.pf}, \mathsf{EO.ek}) \leftarrow \mathsf{EO.EKG}$. The simulator defines $\{\mathsf{h}_{C,i,j}\}_{i\in[m_C],j\in[m_{C,i}]}$ and $\{\mathsf{h}_{i,j}\}_{i\in[M],j\in[m_i]}$ such that they are consistent with the leakage graph. It then defines $\mathsf{EO.ect}_{i,j} \leftarrow \mathsf{EO.Sim}(\mathsf{EO.pk}, \mathsf{EO.sk}, \mathsf{h}_{i,j})$. It samples $\overline{\mathsf{sh}}_{D,i} \xleftarrow{R} \{0,1\}^{|\mathsf{v}|}$ and $(\mathsf{KEM.cp}_i, \mathsf{KEM.k}_i) \leftarrow \mathsf{KEM.Enc}(\mathsf{KEM.pk})$. Define $\widetilde{\mathsf{sh}}_{D,i}$ s.t. that it is consistent with SH_D and $\overline{\mathsf{sh}}_{D,i}$. For all $\widetilde{\mathsf{sh}}_{D,i}$ that are not defined yet, sample $\widetilde{\mathsf{sh}}_{D,i} \xleftarrow{R} \{0,1\}^{|\mathsf{v}|}$. For $t \notin \mathbb{C}$, it samples $\mathsf{seed}_t \xleftarrow{R} \{0,1\}^\kappa$ and $\mathsf{cta}_t \leftarrow \mathsf{PKE.Enc}(\mathsf{pk}, \mathsf{seed}_t)$, which is identical to the protocol description.

We prove the claim by using the following sequence of hybrids.

Hybrid_0: Is identical to the views during the real execution of the protocol.
Hybrid_1: Simulates the view during the shuffling by using the simulator of the shuffle protocol.
Hybrid_2: Sample $\mathsf{EO.ect}_{i,j} \leftarrow \mathsf{EO.Sim}(\mathsf{EO.pk}, \mathsf{EO.sk}, \mathsf{h}_{i,j})$, where $\mathsf{h}_{i,j} := \mathsf{EO.Eval}(\mathsf{EO.ek}, \mathsf{p}_{i,j})$ and $\mathsf{p}_{i,j}$ is the reshuffled $\mathsf{p}_{t,i,j}$, which can be computed from the shuffle permutation π and $\{\mathsf{KV}_t\}_{t\in[T]}$.
Hybrid_3: It defines $\{\mathsf{h}_{C,i,j}\}_{i\in[m_C],j\in[m_{C,i}]}$ and $\{\mathsf{h}_{i,j}\}_{i\in[M],j\in[m_i]}$ such that they are consistent with the leakage graph.
Hybrid_4: Samples $(\mathsf{KEM.cp}_i, \mathsf{KEM.k}_i) \leftarrow \mathsf{KEM.Enc}(\mathsf{KEM.pk})$ s.t. it is independent of $\overline{\mathsf{sh}}_{D,i}$, $\widetilde{\mathsf{sh}}_{D,i}$ and SH_D.
Hybrid_5: Samples $\overline{\mathsf{sh}}_{D,i} \xleftarrow{R} \{0,1\}^{|\mathsf{v}|}$ and defines $\widetilde{\mathsf{sh}}_{D,i}$ s.t. that it is consistent with $\mathsf{SH}_{\mathcal{J},D}$ and $\overline{\mathsf{sh}}_{D,i}$. For all $\widetilde{\mathsf{sh}}_{D,i}$ that are not defined yet, sample $\widetilde{\mathsf{sh}}_{D,i} \xleftarrow{R} \{0,1\}^{|\mathsf{v}|}$.

If Hybrid_0 and Hybrid_1 can be distinguished with probability ϵ, then there is a distinguisher against the simulator of the shuffle protocol with probability ϵ. Since such a distinguishing probability is negligible (based on the security of EO, see Theorem 4), distinguishing Hybrid_0 from Hybrid_1 is also negligible.

If Hybrid_1 and Hybrid_2 can be distinguished with probability ϵ, we can define a sequence of hybrids that step by step replaces $\mathsf{EO.ect}_{i,j}$ with outputs of $\mathsf{EO.Sim}$. Now, there are at least two consecutive intermediate hybrids that can be distinguished with probability $\epsilon/(\sum_{i=1}^{M} m_i)$. Since this probability is negligible due to the evaluated ciphertext simulatability of EO, Hybrid_1 and Hybrid_2 can also only be distinguished with negligible probability.

In Hybrid_2 $\mathsf{h}_{i,j}$ and $\mathsf{h}_{C,i,j}$ are the outputs of $\mathsf{EO.Eval}$ whereas in Hybrid_3 they are uniform in $\{0,1\}^\kappa$. We prove that Hybrid_2 and Hybrid_3 are indistinguishable except with negligible probability by a reduction to the pseudorandomness of $\mathsf{EO.Eval}$. Let there be a distinguisher distinguishing Hybrid_2 and Hybrid_3 with probability ϵ, then we can build a distinguisher against the pseudorandomness of $\mathsf{EO.Eval}$ with probability ϵ. The latter requests all $\mathsf{h}_{i,j}$ and $\mathsf{h}_{C,i,j}$ from the $\mathsf{EO.Eval}$ oracle, uses them to simulate Hybrid_2, Hybrid_3 and outputs the output of the former distinguisher. When they are actual $\mathsf{EO.Eval}$ outputs, it simulates Hybrid_2 and when they are uniform, it simulates Hybrid_3.

If Hybrid_3 and Hybrid_4 can be distinguished with probability ϵ, we can define a sequence of hybrids that step by step replaces $(\mathsf{KEM.cp}_i, \mathsf{KEM.k}_i)$ with $(\mathsf{KEM.cp}_i, \mathsf{KEM.k}'_i)$ where $(\mathsf{KEM.cp}_i, \mathsf{KEM.k}_i) \leftarrow \mathsf{KEM.Enc}(\mathsf{pk})$.

Now there exist two consecutive intermediate hybrids that can be distinguished which implies a distinguisher for $(\mathsf{KEM.cp}, \mathsf{KEM.k})$ and $\mathsf{KEM.cp}, \mathsf{KEM.k}'$ with probability ϵ/M. By the triangular inequality, we can then build a distinguisher for $(\mathsf{KEM.cp}, \mathsf{KEM.k})$ and $(\mathsf{KEM.cp}, u)$ with probability $\epsilon/2$, where $u \xleftarrow{R} \{0,1\}^{|\mathsf{v}|}$. Such a distinguisher breaks the key indistinguishability for the KEM. Since this is negligible, Hybrid_3 and Hybrid_4 cannot be distinguished except with negligible probability.

Hybrid_4 and Hybrid_5 produce identically distributed views. Note that $\widetilde{\mathsf{sh}}_{D,i}$, $\widetilde{\mathsf{sh}}_{D,i}$ and $\mathsf{SH}_{\mathcal{J},D}$ are independent of $(\mathsf{KEM.cp}_i, \mathsf{KEM.k}_i)$. Further, due to the simulator of the shuffling, they are independent of $\mathsf{sh}_{C,t,i}$ and $\mathsf{sh}_{D,t,i}$. Therefore, they can be sampled independently which concludes the proof of our claim. □

Claim. Let EO be a secure and correct randomizable encrypted OPRF scheme. Then, there exists a simulator that generates the joint view of Party S and any subset of parties P_1 to P_T that is indistinguishable from the joint view during the protocol execution.

Proof. The joint view of Party S and the corrupted subset of parties P_1 to P_T (defined by set $\mathbb{C} \subseteq [T]$) can be generated from their input $\{\mathsf{KV}_t\}_{t\in\mathbb{C}}$ and the received messages $\{\widetilde{\mathsf{sh}}_{C,i}\}_{i\in[M]}$ and $\mathsf{KEM.pk}$.

The simulator samples $\widetilde{\mathsf{sh}}_{C,i} \xleftarrow{R} \{0,1\}$ and uses the simulator of the shuffle protocol to simulate the view during the shuffling.

The view generated by the simulator is indistinguishable from the view during the real protocols by the indistinguishability of the simulated view of shuffling from the real view of the shuffling. Notice that in case of using the simulated view of the shuffling, $\{\widetilde{\mathsf{sh}}_{C,i}\}_{i\in[M]}$ are independent of $\{\mathsf{sh}_{C,t,i}\}_{i\in[m_t],t\in[T]}$. Therefore, $\widetilde{\mathsf{sh}}_{C,i}$ can be sampled independently when using the simulated view during the shuffling. □

□

E Extending Left Join to Inner Join

DPMC and $\mathsf{D}_s\mathsf{PMC}$ can be extended to support other types of joins such as an inner join instead of a left join. In both protocols, party D performs the join based on the encrypted datasets of C and all delegators (i.e., in DPMC in step ④ and in $\mathsf{D}_s\mathsf{PMC}$ in step ⑨). Performing the left join in party D hides from party C which of its rows have been matched with one of the delegators' rows and which have not. It is straightforward to extend our delegated protocols to compute the inner join (i.e., intersection) between KV_C and KV_P and secret share the associated metadata for these rows. This can be performed very efficiently using hash join over the encrypted identifiers and sending the $\widehat{\mathsf{KEM.cp}}$ value to C only for the records present in both datasets. Notably, computing the inner join leaks the intersection size to party C but also renders the downstream MPC computation more efficient since it does not have to process secret shares of `NULL`.