Private Set Operations from Multi-Query Reverse Private Membership Test

Yu Chen [*] Min Zhang[*] Cong Zhang [†] Minglang Dong[*]

Abstract

Private set operations allow two parties perform secure computation on two private sets, such as intersection or union related functions. In this paper, we identify a framework for performing private set operations. At the technical core of our framework is multi-query reverse private membership test (mqRPMT), which is a natural extension of RPMT recently proposed by Kolesnikov et al. [KRTW19]. In mqRPMT, a client with set $X = (x_1, \dots, x_n)$ interacts with a server holding a set Y. As a result, the server only learns a bit vector $(e_1, \dots, e_n)$ indicating whether $x_i \in Y$ but without knowing the value of x_i, while the client learns nothing. We present two constructions of mqRPMT from newly introduced cryptographic primitive and protocol. One is based on commutative weak pseudorandom function (cwPRF), the other is based on permuted oblivious pseudorandom functions (pOPRF). Both cwPRF and pOPRF can be instantiated from the decisional Diffie-Hellman like assumptions in the random oracle model. We also introduce a slight weak version of mqRPMT dubbed mqRPMT*, in which the client learns the cardinality of $X \cap Y$. We show mqRPMT* can be build from a category of multi-query private membership test (mqPMT) called Sigma-mqPMT, which in turn can be realized from DDH-like assumptions or oblivious polynomial evaluation. This makes the first step towards establishing the relation between mqPMT and mqRPMT.

We demonstrate the practicality of our framework with implementations. By plugging our cwPRF-based mqRPMT to the general framework, we obtain various efficient PSO protocols that are competitive or superior to the-state-of-art protocols. For cardinality functionality, our protocol achieves a $1.17 - 6.62\times$ speedup in running time and $10.85 - 14.80\times$ shrinking in communication cost. For cardinality-with-sum functionality, our protocol achieves a $8 - 40\times$ speedup in running time. For union functionality, our protocol achieves strict linear complexity. Among all the existing PSU protocols, it requires the least concrete communication cost, and is also the fastest one in the WAN setting. Specifically, for input set of size 2^{20}, our PSU protocol requires roughly 100 MB bandwidth, and 58 seconds using 4 threads in the LAN setting. For private-ID functionality, our protocol achieves a $1.39 - 4.75\times$ speedup in running time. Moreover, by plugging our FHE-based mqRPMT* to the general framework, we obtain a PSU* protocol (the sender additionally learns the intersection size) suitable for unbalanced setting, whose communication complexity is linear in the size of the smaller set, and logarithmic in the larger set.

Keywords: PSO, PSU, multi-query RPMT, commutative weak PRF, permuted OPRF

[*]Shandong University. Email: yuchen.prc@gmail.com, {zm_min, minglang_dong}@mail.sdu.edu.cn
[†]SKLOIS, IIE, Chinese Academy of Sciences. Email: zhangcong@iie.ac.cn

Contents

1 Introduction

Consider two parties, each with a private set of items, want to compute on their respect sets without revealing any other information to each other. Two-party private set operation (PSO) refers to such family of interactive cryptographic protocols that takes two private sets as input, computes the desired function, and outputs the result to one or both of the participants. If one party obtains the result, we call this party the receiver and the other party the sender, and refer to the protocol as one-sided. Two-sided protocol in the semi-honest setting can be realized by having the receiver in one-sided protocol forwards the result to the sender. In what follows, we briefly introduce PSO protocols in the semi-honest model in terms of typical functionalities.

Private set intersection. PSI has found many applications including privacy-preserving sharing, private contact discovery, DNA testing, pattern matching and so on. Due to its importance and wide applications, in the past two decades PSI has been extensively studied in a long sequence of works and has become truly practical with extremely fast implementation. We refer to [PSZ18] for a good survey of different PSI paradigms. State-of-the-art two-party PSI protocols [KKRT16, PRTY19, CM20, GPR+21, RS21] mainly rely on symmetric-key operations, except a little public-key operations in base OT used in the OT extension protocol.

Private computing on set intersection. Certain real-world application scenarios only requires partial/aggregated information about the intersection. In this setting a more fine-grained private computation on set intersection (PCSI) is needed, including PSI-card for intersection cardinality [HFH99], PSI-card-sum for intersection cardinality and sum [IKN+20, GMR+21].

Private set union. Like PSI, PSU also has numerous applications in practice, such as cyber risk assessment and management via joint IP blacklists and joint vulnerability data. According to the underlying cryptographic techniques, existing PSU protocols can be roughly divided into two categories. The first is mainly based on public-key techniques [KS05, Fri07, HN10, DC17]. The second is mainly based on symmetric-key techniques [KRTW19, GMR+21].

PSO protocols are primarily designed for the balanced setting, namely, the two sets size are approximately the same. Rcently, some works consider unbalanced setting, in which one set is much more larger than the other. PSI, PCSI and PSU are closely related functionalities. Among them, PSI has been extensively studied. The state-of-the-art PSI is almost as efficient as the naive insecure hash-based protocol. In contrast to the affairs of PSI, the efficiency of PCSI and PSU are less satisfactory. In balanced setting, there are PSI protocols [Mea86, CM20] and PCSI protocols [IKN+20] with optimal linear complexity. In unbalanced setting, there are PSI protocols [CLR17, CHLR18, CMdG+21] with sublinear complexity in the larger set size, but no such PCSI protocol is known. For a long time being, there is no PSU protocol with linear complexity in either balanced or unbalanced setting in the literature. It is until very recently, Zhang et al. [ZCL+22] make a breakthrough by proposing the first PSU with linear complexity in the balanced setting. As to practical efficiency, PSI-card is concretely about 20$\times$ slower and requires over 30$\times$ more communication than PSI, and PSU is concretely about 20$\times$ slower and requires over 25$\times$ more communication than PSI.

It is somewhat surprising that the state-of-the-art protocols for different functionalities have significantly different efficiency. Why is this case? Observe that PSI protocol essentially can be viewed as multi-query private membership test (mqPMT), which has very efficient realizations in both balanced and unbalanced setting. However, mqPMT generally does not implies PCSI or PSU. The reason is that mqPMT reveals information about intersection, which should be hidden from the receiver in PCSI and PSU.

1.1 Motivation

Our motivation of this work is three-folds. First, the above discussion indicates that the most efficient PSI protocols may not be easily adapted to PCSI and PSU protocols. Therefore, different approaches are employed for different private set operations, creating much more engineering effort. We are motivated to seek for a common core protocol that enables all private set operations, with the hope to design PSO in a unified framework. Second, given the huge efficiency gap between PSI and other closely related protocols, we are also motivated to give efficient instantiations of the framework to close the gap. Last but not the least, recall that the seminal PSI protocol [Mea86] (related ideas were appeared in [Sha80, HFH99]) is based on the decisional Diffie-Hellman assumption, known as DH-PSI. After roughly four decades, DH-PSI is still the most easily understood and to implement one among numerous PSI protocols. Somewhat surprisingly, no counterpart is known in the PSU setting yet. It is interesting to know if the DDH assumption can strike back. In summary, we are intriguing to know:

Is there a central building block that enables a unified framework for all private set operations mentioned above? If so, can we give efficient instantiations with optimal asymptotic complexity and good concrete efficiency? Can the DDH assumption strike back with efficient PSU protocol?

1.2 Our Contribution

In this work, we make positive answers to the aforementioned questions. We summarize our contribution as below.

A framework of PSO. We identify that multi-query reverse private membership test (mqRPMT) is a "Swiss Army Knife" for various private set operations. In a nutshell, mqRPMT is a two-party protocol between a client holding set $X = (x_1, \dots, x_n)$ and a server holding a set Y. After execution of the protocol, the server learns an indication bit vector $(e_1, \dots, e_n)$ such that $e_i = 1$ if and only if $x_i \in Y$ but without knowing x_i, while the client learns nothing. mqRPMT itself already implies PSI-card; by coupling with OT, mqRPMT implies PSI and PSU; by further coupling with masking technique or AHE, mqRPMT implies PSI-sum-card. Therefore, mqRPMT enables a unifying PSO framework, which can perform a variety of private set operations in a flexible manner.

Efficient construction of mqRPMT. We propose two generic constructions of mqRPMT. The first is based on a new cryptographic primitive called commutative weak PRF (cwPRF), while the second is based on another new secure protocol called permuted oblivious PRF (pOPRF). Both of them can be realized from DDH-like assumptions in the random oracle model, yielding incredibly simple mqRPMT protocols with linear communication and computation complexity. Note that the asymptotic complexity of our PSO framework is dominated by the underlying mqRPMT. Therefore, all PSO protocols derived from our framework inherit same linear complexity. Particularly, we obtain a PSU protocol with optimal linear complexity, which is arguably the most simple and efficient one among existing protocols.

Connection to mqPMT. mqRPMT is of great theoretical interest since it is the core building block of the PSO framework. It is thus interesting to investigate the relation between mqRPMT and mqPMT. Towards this goal, we put forward a variant of mqRPMT called mqRPMT with cardinality (denoted by mqRPMT* hereafter). Compared to the standard mqRPMT, mqRPMT* additionally reveals the intersection size to the client. We show that mqRPMT* can be built from a broad class of mqPMT called Sigma-mqPMT in a black-box manner via the "permute-then-test" approach. This makes the initial step towards to establishing the connection between mqRPMT and mqPMT. We argue that though mqRPMT* deviates from standard mqRPMT in revealing additional information (intersection size) to the client, it could also be a desirable feature in application scenarios where both parties want to learn intersection size, for example, PSI-card-sum [IKN+20]. We leave the general connection between mqPMT and mqRPMT as a challenging open problem.

Evaluations. We give efficient instantiation of our generic framework from cwPRF-based mqRPMT protocols. We provide C++ implementations. The experimental results demonstrate that except PSI, most PSO protocols derived from our generic framework is competitive or superior to the state-of-the-art corresponding protocols.

1.3 Technical Overview

PSO from mqRPMT. As discussed above, mqPMT (a.k.a. PSI) protocol generally is not applicable for computing PCSI and PSU. We examine the reverse direction, i.e., whether the core protocol underlying PSU can be used for computing PSI and PCSI. We identify that the central protocol beneath all the existing PSU protocols is mqRPMT, which is a generalization of RPMT formalized in [KRTW19]. Roughly speaking, mqRPMT is a two-party protocol between a client with set $X = (x_1, \dots, x_n)$ and a server with set Y. After execution of the protocol, the server learns an indication bit vector $(e_1, \dots, e_n)$ such that

$e_i = 1$ if and only if $x_i \in Y$ but without knowing x_i, while the client learns nothing. Superficially, mqRPMT is similar to mqPMT, except that the server but not the client learns the test results instead. This subtle difference turns out to be significant. To see this, note that in mqRPMT the intersection (except its cardinality) is hidden from both sides, while in mqPMT the intersection is finally known by the client. In light of this difference, mqRPMT is particular suitable for functionalities that have to keep intersection private. With mqRPMT in hand, a PSU protocol is immediate by having the sender (play the role of client) and the receiver (play the role of server) invoke a mqRPMT protocol on the first place, then carrying out n one-sided OT with e_i and y_i respectively. PSI and other protocols such as PSI-card and PSI-card-sum can be constructed similarly by coupling with simple masking technique or additively homomorphic encryption.

Next, we show two generic constructions of mqRPMT.

mqRPMT from cwPRF. We propose a new cryptographic primitive called commutative weak PRF. Let $F : K \times D \to R$ be a family of weak PRF, where $R \subseteq D$. We say F is commutative if for any $k_1, k_2 \in K$ and any $x \in D$, it holds that $F_{k_1}(F_{k_2}(x)) = F_{k_2}(F_{k_1}(x))$. In other words, the two composite functions $F_{k_1} \circ F_{k_2}$ and $F_{k_2} \circ F_{k_1}$ are essentially the same function, say, $\hat{F}$.

We then show how to build mqRPMT from cwPRF. Let server and client generate cwPRF key k_1 and k_2 respectively, and both of them map their items to elements in the domain D of F via a common cryptographic hash function H, which will be modeled as a random oracle. We begin with the construction of the basic single-query RPMT. First observe that cwPRF gives rise to a simple private equality test (PEQT) protocol. More precisely, a server P_1 holding y and client P_2 holding x can conduct PEQT functionality via the following steps: (1) P_1 computes and sends $F_{k_1}(\mathsf{H}(y))$ to P_2; (2) P_2 computes and sends $F_{k_2}(\mathsf{H}(x))$ and $F_{k_2}(F_{k_1}(\mathsf{H}(y)))$ to P_1; (3) P_1 then learns the test result by comparing $F_{k_1}(F_{k_2}(\mathsf{H}(x))) =? F_{k_2}(F_{k_1}(\mathsf{H}(y)))$. The commutative property of F ensures the correctness. The weak pseudorandomness of F guarantees that P_2 learns nothing and P_1 learns nothing beyond the test result. At a high level, $F_{k_2}(F_{k_1}(\mathsf{H}(\cdot))) = F_{k_1}(F_{k_2}(\mathsf{H}(\cdot))) = \hat{F}_k(\mathsf{H}(\cdot))$ serves as pseudorandom encoding function in the joint view, while $F_{k_1}(\mathsf{H}(\cdot))$ and $F_{k_2}(\mathsf{H}(\cdot))$ serve as a partial encoding function in the individual views of server and client respectively.

We observe that PEQT not only can be viewed as an extreme case of mqPMT [PSZ14], but can also be viewed as an extreme case of mqRPMT (by degrading the sets on both sides to singletons). Therefore, we choose PEQT as starting point to build mqRPMT. We first extend the elements on server side from a singleton to a set. However, naive repetition of the above PEQT protocol by sending back $F_{k_2}(F_{k_1}(\mathsf{H}(y_i)))$ for each $y_i \in Y$ in the same order of server's first move message $F_{k_1}(\mathsf{H}(y_i))$ does not lead to a secure single-query mqRPMT. The reason is that $\{\hat{F}_k(\mathsf{H}(y_i))\}_{i \in [n]}$ constitutes an order preserving pseudorandom encoding of set Y. As a consequence, the server will learn the exact value of x if $x \in Y$. In order to perform the membership test in an oblivious manner, the idea is to make the pseudorandom encoding of Y independent of the order known by the server. A straightforward approach is to shuffle $\{\hat{F}_k(\mathsf{H}(y_i))\}$. In this way, we obtain a single-query RPMT protocol from cwPRF, and the resulting protocol can be easily batched to handle multiple queries by reusing the pseudorandom encoding of X. Suppose the set size on both sides is n. A simple calculation shows that the computation cost is $3n$ times evaluation of F and n times look up, and the communication cost is $4n$ elements in the range of F. The mqRPMT protocol is optimal in the sense that both computation and communication complexity are linear to the set size. We can further reduce the communication cost by inserting $\{\hat{F}(\mathsf{H}(y_i))\}$ into an order hiding data structure such as Bloom filter, instead of permuting them.

We show that cwPRF can be realized from DDH-like assumptions. Henceforth, DDH strikes back with an incredibly simple PSU protocol. This once again demonstrates that the DDH assumption is truly a golden goose in cryptography.

mqRPMT from permuted OPRF. OPRF provides a conceptually simple approach to build PSI (a.k.a. mqPMT) protocols. We exemplify this by recalling the multi-point OPRF-based PSI [PRTY19, CM20] as below: the sender with Y and the receiver with $X = (x_1, \ldots, x_n)$ first engage in a multi-point OPRF protocol. As a result, the sender obtains a random key k of PRF F, while the receiver obtains PRF values $(F_k(x_1), \ldots, F_k(x_n))$. Subsequently, the sender randomly shuffles the elements in his own set, obtains $(y_1, \ldots, y_n)$, and sends the corresponding PRF values $(F_k(y_1), \ldots, F_k(y_n))$ to the receiver. Finally, the receiver obtains the intersection by checking if $F_k(x_i) \in \{F_k(y_i)\}_{i \in [n]}$ for each $x_i \in X$. It is interesting to investigate if the above instructive approach can also be used to compute PSU. However, OPRF does not readily imply a mqRPMT protocol. The reason is that the receiver learns the PRF values with the same order of his input $X = (x_1, \ldots, x_n)$. To remedy this problem, we introduce a new cryptographic protocol called permuted OPRF (pOPRF). pOPRF can be viewed as a generalization of OPRF. The difference is that the sender additionally obtains a random permutation π over $[n]$ besides PRF key k, while the receiver obtains PRF values in a permuted order as per π. pOPRF immediately implies a mqRPMT protocol: The server with $Y = (y_1, \ldots, y_n)$ and the client with $X = (x_1, \ldots, x_n)$ first engage in a pOPRF protocol. As a result, the server obtains $\{F_k(y_{\pi(i)})\}_{i\in[n]}$, while the client learns a PRF key k and a permutation π. The client then computes and sends $\{F_k(x_i)\}_{i\in[n]}$. Finally, the server learns if $x_i \in Y$ by testing whether $F_k(x_i) \in \{F_k(y_{\pi(i)})\}_{i\in[n]}$, but learns nothing more since its PRF values are of permuted order. At a high level, $F_k(\cdot)$ serves as an encoding function in client's view, while $F_k(\pi(\cdot))$ serves as a pseudorandom and permuted encoding function in server's view.

The question remains is how to build pOPRF. One common approach to build OPRF is "mask-then-unmask". We choose OPRF as the starting point. The rough idea is exploiting the input homomorphism to mask inputs[1], then unmask the outputs. If the mask procedure is different per input, then the unmask procedure must be carried out accordingly. Therefore, OPRF protocols of this case cannot be easily adapted to pOPRF, since the receiver is unable to perform the unmask procedure over permuted masked outputs correctly, namely, recovering outputs in permuted order. The above analysis indicates us that if the masking procedure can be done via a unifying manner, then the receiver might be able to unmask the permuted masked outputs correctly. Observe that the simplest way to perform unified masking is to apply a weak pseudorandom function G_s to the intermediate input $\mathsf{H}(x)$. To enable efficient unmask procedure, we further require that G_s is a permutation and commutative with respect to F_k. This yields a simple pOPRF construction from enhanced cwPRF (F_k, G_s) in which $G_s(\cdot)$ is a weak pseudorandom permutation. More precisely, to build pOPRF, the sender picks a random PRF key k for F, while the receiver with input $X = (x_1, \ldots, x_n)$ picks a random PRP key s for G. The receiver then sends $\{G_s(\mathsf{H}(x_i))\}_{i\in[n]}$ to the sender. Upon receiving the masked intermediate inputs, the sender applies F_k to them, then sends the results in permuted order, a.k.a. $\{F_k(G_s(\mathsf{H}(x_{\pi(i)})))\}_{i\in[n]}$. Finally, the receiver applies G_s^{-1} to the permuted masked outputs, and will obtain $\{F_k(\mathsf{H}(x_{\pi(i)}))\}_{i\in[n]}$ by the commutative property.

Note that many efficient OPRF constructions [CM20] seem not amenable to pOPRF construction due to lack of nice algebra structures. This somehow explains the efficiency gap between the state-of-the-art PSI and PCSI/PSU.

mqRPMT* from Sigma-mqPMT. Towards the goal of studying the connection between mqRPMT and mqPMT, we first abstract a category of mqPMT protocols called Sigma-mqPMT. The starting point of Sigma-mqPMT is Sigma-PMT. Roughly speaking, Sigma-PMT is a three-move protocol, which proceeds as below: (1) in the first move, the server holding a set Y sends a message a to the client, where a is best interpreted as an encoding of Y; (2) in the second move, the client makes a test query q of its item x; (3) in the last move, the server responds with z, and eventually the client can decide if $x \in Y$ by running algorithm $\mathsf{Test}(a, q, x, z)$. To enable efficient parallel composition, we introduce the following two properties for Sigma-PMT: (i) reusable property, which ensures the first move message can be safely reused over multi-instance; (ii) context-independent property, which means the test query only depends on the item in test. With these two properties, one can build mqPMT by running multiple instances of Sigma-PMT in parallel, without increasing round complexity. If the underlying Sigma-PMT additionally satisfies stateless testing, namely, Test algorithm can be done without learning (q, x), we refer to the resulting mqPMT as Sigma-mqPMT, which captures the common form of several PSI protocols [Mea86, FIPR05, CLR17]. By utilizing the stateless property, we can tweak Sigma-mqPMT to permuted mqPMT via the permute-then-test approach, without incurring computation and communication overhead, while permuted mqPMT instantly implies mqRPMT*. Therefore, we can expand a series of results in PSI setting to PSO setting, on the premise that revealing intersection size is acceptable. Notably, by applying the conversion to fully homomorphic encryption (FHE) based Sigma-mqPMT, we obtain an efficient mqRPMT* in unbalanced setting, which gives rise to the first PSU* protocol whose communication complexity is sublinear to the size of larger set X.

In Figure 1, we give an overview of the main contribution of this work.

[1]Standard pseudorandomness denies input homomorphism. Rigorously speaking, we utilize the homomorphism over intermediate input.

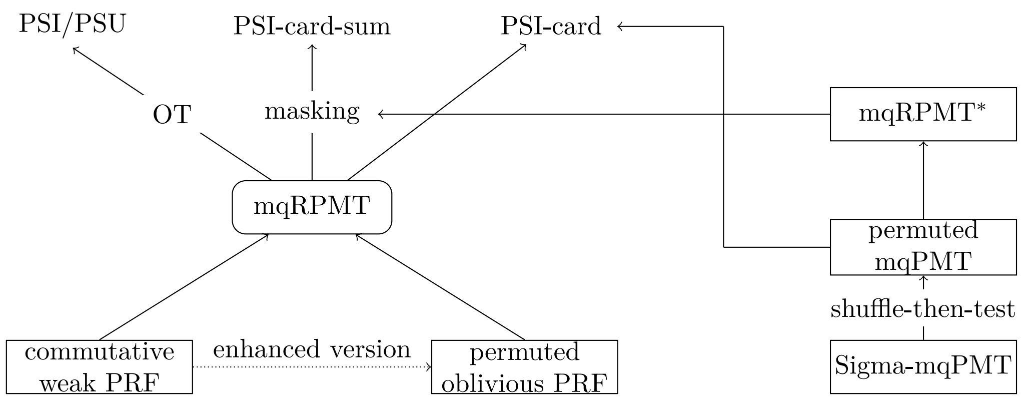


Figure 1: An overview of our main results. The rounded rectangles denote our contributions. The rectangles denotes notions in previous works.

1.4 Related Works

Ion et al. [IKN+20] show how to transform single-point OPRF-based [PSZ14, KKRT16], garbled Bloom filter-based [DCW13, RR17], and DDH-based [HFH99] PSI protocols into one for computing PSI-card-sum by leveraging additively homomorphic encryption (AHE). However, their conversions are not pretty efficient due to the usage of AHE, and as noted by the authors, detailed conversions to each category of protocols differ significantly, especially in the way of making use of the underlying AHE. In contrast to their work, we distill a broad class of PSI protocols as Sigma-mqPMT, then show how to tweak it to mqRPMT* in a generic and black-box manner, without relying on any additional cryptographic tools. Our conversion works in a more abstract and lower level, and such generality lends it may find more potential applications. Miao et al. [MPR+20] put forwards shuffled distributed oblivious PRF as a central tool to build PSI-card-sum with malicious security. Compared to shuffled distributed OPRF, our notion of permuted OPRF is much simpler and should be best viewed as a useful extension of standard OPRF. The conceptual simplicity lends it can be easily built from enhanced cwPRF and find more potential applications. For example, permuted OPRF immediately implies permuted multi-point private equality test, which is a key tool in building FHE-based PSU [TCLZ22]. Garimella et al. [GMR+21] propose a framework for all private set operations. At their technical core is a new protocol called permuted characteristic, which could be viewed as an extension of mqRPMT protocol. Nevertheless, the oblivious shuffle in permuted characteristic functionality is not necessary for PSO, but seems unavoidable due to the use of oblivious switching networks. This incurs superlinear complexity to permuted characteristic protocol and the enabling PSO protocols. Moreover, we note that the PSI-card-sum functionality defined in [GMR+21] differs from from the original functionality defined in [IKN+20]. The distinction is that in the original functionality of PSI-card-sum, both parties are given the cardinality of intersection, and the party initially holding values is also given the intersection sum, while in the functionality described in [GMR+21], it is the party without holding values who is given the cardinality and sum of intersection. To highlight this subtle difference, we prefer to call the functionality presented in [GMR+21] as reverse PSI-card-sum.

Concurrent work. Very recently, Zhang et al. [ZCL+22] propose a generic construction of mqRPMT with linear complexity from oblivious key-value store, set-membership encryption and oblivious vector decryption-then-test functionality. By instantiating their generic construction from public-key and symmetric-key encryption respectively and combining OT, they make the breakthrough by giving the first PSU protocol with optimal linear complexity. However, as noted by the authors, the more efficient PKE-based construction is leaky, failing to satisfy the standard security of mqRPMT. Compared to their work, our construction of mqRPMT is much simpler and realization meets the standard definition. Besides, we explore mqRPMT as a central building block for a family of private set operations, while their main focus is limited to PSU.

2 Preliminaries

2.1 Notation

We use κ and λ to denote the computational and statistical parameter respectively. Let $\mathbb{Z}_n$ be the set $\{0, 1, \ldots, n-1\}$, $\mathbb{Z}_n^* = \{x \in \mathbb{Z}_n \mid \gcd(x, n) = 1\}$. We use $[n]$ to denote the set $\{1, \ldots, n\}$. We assume that every set X has a default order (e.g. lexicographical order), and write it as $X = \{x_1, \ldots, x_n\}$. For a set X, we use $|X|$ to denote its size and use $x \xleftarrow{\text{R}} X$ to denote sampling x uniformly at random from X. We use $(x_1, \ldots, x_n)$ to denote a vector, whose ith element is x_i. A function is negligible in κ, written $\mathsf{negl}(\kappa)$, if it vanishes faster than the inverse of any polynomial in κ. A probabilistic polynomial time (PPT) algorithm is a randomized algorithm that runs in polynomial time.

2.2 MPC in the Semi-honest Model

We use the standard notion of security in the presence of semi-honest adversaries. Let Π be a protocol for computing the function $f(x_1, x_2)$, where party P_i has input x_i. We define security in the following way. For each party P, let $\text{View}_P(x_1, x_2)$ denote the view of party P during an honest execution of Π on inputs x_1 and x_2. The view consists of P's input, random tape, and all messages exchanged as part of the Π protocol.

Definition 2.1. 2-party protocol Π securely realizes f in the presence of semi-honest adversaries if there exists a simulator Sim such that for all inputs x_1, x_2 and all $i \in \{1, 2\}$:

$$\mathsf{Sim}(i, x_i, f(x_1, x_2)) \approx_c \text{View}_{P_i}(x_1, x_2)$$

Roughly speaking, a protocol is secure if the party with x_i learns no more information other than $f(x_1, x_2)$ and x_i.

2.3 Private Set Operation

PSO is a special case of secure two-party computation. We call the two parties engaging in PSO the *sender* and the *receiver*. The sender holds a set X of size n_x, and the receiver holds a set Y of size n_y (for simplicity, we assume $n_\mathsf{x} = n_\mathsf{y} = n$). Both sets consist of ℓ-bit strings. We always assume the set sizes n_x and n_y are public. The ideal PSO functionality (depicted in Figure 2) computes the intersection, union, cardinality, or intersection sum, outputs nothing to the sender, and $X \cup Y$ to the receiver.

Parameters: size of sets n.

Functionality: On input $X = \{x_1, \ldots, x_n\} \subseteq \{0,1\}^\ell$ from the sender P_1 and $Y = \{y_1, \ldots, y_n\} \subseteq \{0,1\}^\ell$ (and possibly $V = \{v_1, \ldots, v_n\}$) from the receiver P_2:

- **intersection:** give $X \cap Y$ to the receiver P_2.
- **union:** give $X \cup Y$ to the receiver P_2.
- **union*:** give $|X \cap Y|$ to the sender P_1 and $X \cup Y$ to the receiver P_2.
- **card:** give $|X \cap Y|$ to the receiver P_2.
- **card-sum:** give $|X \cap Y|$ to both the sender P_1 and the receiver P_2, and give $S = \sum_{i: x_i \in Y} v_i$ to the receiver P_2.

Figure 2: Ideal functionality $\mathcal{F}_{\mathsf{PSO}}$ for PSO

3 Protocol Building Blocks

3.1 Oblivious Transfer

Oblivious Transfer (OT) [Rab] is a central cryptographic primitive in the area of secure computation. 1-out-of-2 OT allows a sender with two input strings (m_0, m_1) and a receiver with an input choice bit $b \in \{0, 1\}$. As a result of the OT protocol, the receiver learns m_b and neither party learns any additional information. Though expensive public-key operations is unavoidable for a single OT, a powerful technique called OT extension [IKNP03, KK13, ALSZ15] allows one to perform n OTs by only performing $O(\kappa)$ public-key operations (where κ is the computational security parameter) and $O(n)$ fast symmetric-key operations. In Figure 3 we formally define the ideal functionality for OT that provides n parallel instances of OT.

Parameters: number of OT instances n; string length ℓ.

Functionality: On input $\{(m_{i,0}, m_{i,1})\}_{i \in n}$ from the sender P_1 where each $m_{i,b} \in \{0,1\}^\ell$, and input $\vec{b} \in \{0,1\}^n$ from the receiver P_2:

- Give output $(m_{1,b_1}, \ldots, m_{n,b_n})$ to the receiver.

Figure 3: Ideal functionality $\mathcal{F}_{\mathsf{OT}}$ for OT

3.2 Multi-Query RPMT

RPMT [KRTW19] refers to a protocol where the client with input x interacts with a server holding a set Y. As a result, the server learns (only) the bit indicating whether $x \in Y$, while the client learns nothing about the set Y. The default notion of RPMT allows the client to query for a single element. While this procedure can be repeated several times, one may seek more efficient solutions allowing the client to make n distinct queries at a reduced cost. This generalized notion of n-time RPMT is straightforward to define. Hereafter, we refer to n-time RPMT as multi-query RPMT. In Figure 4 we formally define the ideal functionality for mqRPMT. We also define a relaxed version of mqRPMT called mqRPMT*, in which the client is given $|X \cap Y|$.

Parameters: number of RPMT queries n.

Functionality: On input set Y from the server P_1 and input set $X = (x_1, \ldots, x_n) \subseteq \{0,1\}^\ell$ from the client P_2:

1. Define the vector $\vec{e} = (e_1, \ldots, e_n) \in \{0,1\}^n$, where $e_i = 1$ if $x_i \in Y$ and $e_i = 0$ otherwise.
2. Give $\vec{e}$ to the server P_1.

Figure 4: Ideal functionality $\mathcal{F}_{\mathsf{mqRPMT}}$ for multi-query RPMT

Family of PMT protocols. For completeness and fixing terminology, we are tempting to systematically list the whole family of PMT protocols. We identify two characteristics of PMT protocols. One is direction, which consists of two options, namely forward or reverse. Standard option means the indication bit indicates the membership of receiver's elements, while reverse option means the indication bit indicates the membership of sender's elements. The other one is order, which also consists of two options, namely ordered and permuted. The ordered option means the indication bit is of the right order (known by the receiver). The permuted option means the indication bit is of the permuted order known by the sender. By mix-match two characteristics, we obtain four types PMT protocols, shown in Table 1.

Table 1: The family of PMT protocols

Protocol	Direction		Order		Direct usage
	forward	reverse	ordered	permuted	
mqPMT	✓		✓		PSI
mqRPMT		✓	✓		PSI-card
permuted mqPMT	✓			✓	PSI-card
permuted mqRPMT		✓		✓	PSI-card

mqPMT and PSI are the same protocol under different names. mqRPMT is formalized in [KRTW19, ZCL+22]. Permuted mqRPMT is introduced in [GMR+21] under the name of permuted characteristic. To the best of our knowledge, the notion of permuted mqPMT is new to this work, which could be viewed as a high-level abstraction of the DDH-based PSI-card protocol due to [HFH99].

4 Review of Pseudorandom Function

In this section, we recap the standard notions of PRF, as well as the canonical construction from the DDH like assumption. Looking ahead, we will build more advanced variants of PRF with richer properties on these basis. We first recall the notion of standard pseudorandom functions (PRFs) [GGM86].

Definition 4.1 (PRF)**.** A family of PRFs consists of three polynomial-time algorithms as follows:

- $\mathsf{Setup}(1^\kappa)$: on input a security parameter κ, outputs public parameter pp. pp specifies a family of keyed functions $F: K \times D \rightarrow R$, where K is the key space, D is domain, and R is range.
- $\mathsf{KeyGen}(pp)$: on input pp, outputs a secret key $k \xleftarrow{\text{R}} K$.
- $\mathsf{Eval}(k, x)$: on input $k \in K$ and $x \in D$, outputs $y \leftarrow F(k, x)$. For notation convenience, we will write $F(k, x)$ as $F_k(x)$ interchangeably.

The standard security requirement for PRFs is pseudorandomness.

Pseudorandomness. Let $\mathcal{A}$ be an adversary against PRFs and define its advantage as:

$$\mathsf{Adv}_{\mathcal{A}}(\kappa) = \Pr\left[\beta' = \beta : \begin{array}{l} pp \leftarrow \mathsf{Setup}(1^\kappa); \\ k \leftarrow \mathsf{KeyGen}(pp); \\ \beta \leftarrow \{0,1\}; \\ \beta' \leftarrow \mathcal{A}^{\mathcal{O}_{\mathsf{ror}}(\beta,\cdot)}(\lambda); \end{array}\right] - \frac{1}{2},$$

where $\mathcal{O}_{\mathsf{ror}}(\beta,\cdot)$ denotes the real-or-random oracle controlled by β, i.e., $\mathcal{O}_{\mathsf{ror}}(0, x) = F_k(x)$, $\mathcal{O}_{\mathsf{ror}}(1, x) = \mathsf{H}(x)$ (here H is chosen uniformly at random from all the functions from D to R[2]). $\mathcal{A}$ can adaptively access the oracle $\mathcal{O}_{\mathsf{ror}}(\beta,\cdot)$ polynomial many times. We say that F is pseudorandom if for any PPT adversary $\mathsf{Adv}_{\mathcal{A}}(\kappa)$ is negligible in κ. We refer to such security as full PRF security.

Sometimes the full PRF security is not needed and it is sufficient if the function cannot be distinguished from a uniform random one when challenged on random inputs. The formalization of such relaxed requirement is *weak pseudorandomness*, which is defined the same way as pseudorandomness except that the inputs of oracle $\mathcal{O}_{\mathsf{ror}}(b,\cdot)$ are uniformly chosen from D by the challenger instead of adversarially chosen by $\mathcal{A}$. PRFs that satisfy weak pseudorandomness are referred to as *weak PRFs*.

4.1 Weak PRF from the DDH Assumption

We recall the folklore weak PRF from the DDH assumption as below.

- $\mathsf{Setup}(1^\kappa)$: runs $\mathsf{GroupGen}(1^\kappa) \rightarrow (\mathbb{G}, g, p)$, outputs $pp = (\mathbb{G}, g, p)$. pp defines a family of functions from $\mathbb{Z}_p \times \mathbb{G}$ to $\mathbb{G}$, a.k.a. on input $k \in \mathbb{Z}_p$ and $x \in \mathbb{G}$ outputs $F_k(x) = x^k$.

[2]To efficiently simulate access to a uniformly random function H from D to R, one may think of a process in which the adversary's queries to $\mathcal{O}_{\mathsf{ror}}(1,\cdot)$ are "lazily" answered with independently and randomly chosen elements in R, while keeping track of the answers so that queries made repeatedly are answered consistently.

- $\mathsf{KeyGen}(pp)$: outputs $k \xleftarrow{\text{R}} \mathbb{Z}_p$.
- $\mathsf{Eval}(k, x)$: on input $k \in \mathbb{Z}_p$ and $x \in D$, outputs $y \leftarrow x^k$.

The following theorem establishes its pseudorandomness based on the DDH assumption.

Theorem 4.1. *$F_k(x)$ is a family of weak pseudorandom functions assuming the hardness the DDH assumption holds w.r.t.* $\mathsf{GroupGen}(1^\kappa) \to (\mathbb{G}, g, p)$.

Proof. DDH assumption states that DDH tuple (g^a, g^b, g^{ab}) and random tuple (g^a, g^b, g^c) are computationally indistinguishable. By exploiting the random self-reducibility of the DDH problem [NR95], the standard DDH assumption implies that $(g^a, g^{b_1}, \ldots, g^{b_n}, g^{ab_1}, \ldots, g^{ab_n})$ and $(g^a, g^{b_1}, \ldots, g^{b_n}, g^{c_1}, \ldots, g^{c_n})$ are computationally indistinguishable, where $a, b_i, c_i \xleftarrow{\text{R}} \mathbb{Z}_p$. We are now ready to reduce the weak pseudorandomness of $F_k(\cdot)$ based on the DDH assumption. Let $\mathcal{B}$ be an adversary against the DDH assumption. Given a DDH challenge instance $(g^a, g^{b_1}, \ldots, g^{b_n}, g^{c_1}, \ldots, g^{c_n})$, $\mathcal{B}$ interacts with an adversary $\mathcal{A}$ in the weak pseudorandomness experiment, with the aim to determine if $c_i = ab_i$ or random values.

Setup: $\mathcal{B}$ sends $pp = (\mathbb{G}, g, p)$ to $\mathcal{A}$. $\mathcal{B}$ implicitly sets a as the key of PRF.

Real-or-random query: Upon receiving the i-th query to oracle $\mathcal{O}_{\mathsf{ror}}$, $\mathcal{B}$ sets the i-th random input $x_i := g^{b_i}$, computes $y_i = g^{c_i}$, then sends (x_i, y_i) to $\mathcal{A}$.

Guess: $\mathcal{A}$ makes a guess $\beta' \in \{0, 1\}$ for β, where '0' indicates real mode and '1' indicates random mode. $\mathcal{B}$ forwards β' to its own challenger.

Clearly, if $c_i = ab_i$ for all $i \in [n]$, then $\mathcal{A}$ simulates the real oracle. If c_i are random values, then $\mathcal{A}$ simulates the random oracle. Thereby, $\mathcal{B}$ breaks the DDH assumption with the same advantage as $\mathcal{A}$ breaks the pseduorandomness of $F_k(\cdot)$. $\square$

Remark 4.1. We note that $F_k(x) = x^k$ is actually a permutation over $\mathbb{G}$, and it is efficiently invertible.

4.2 PRF from the DDH Assumption

We next recall the standard PRF from the DDH assumption known as HashDH presented in [NPR99]. The construction is very similar to the weak PRF construction. The only modification is to map the input to $\mathbb{G}$ via a cryptographic hash function H first, then apply F_k in a cascade way, yielding a composite function $F_k \circ \mathsf{H} : D \to \mathbb{G}$. By leveraging the programmability of H, we reduce to pseudorandomness of the composite function $F_k \circ \mathsf{H}$ to the weak pseudorandomness of F_k. In other words, random oracle amplifies weak pseudorandomness to standard pseudorandomness.

For completeness, we provide the details as below.

- $\mathsf{Setup}(1^\kappa)$: runs $\mathsf{GroupGen}(1^\kappa) \to (\mathbb{G}, g, p)$, picks a cryptographic hash function H from domain D to $\mathbb{G}$, outputs $pp = (\mathbb{G}, g, p, \mathsf{H})$. pp defines a family of functions from $\mathbb{Z}_p \times D$ to $\mathbb{G}$, which takes $k \in \mathbb{Z}_p$ and $x \in D$ as input and outputs $F_k(\mathsf{H}(x)) = \mathsf{H}(x)^k$.
- $\mathsf{KeyGen}(pp)$: outputs $k \xleftarrow{\text{R}} \mathbb{Z}_p$.
- $\mathsf{Eval}(k, x)$: on input $k \in \mathbb{Z}_p$ and $x \in D$, outputs $\mathsf{H}(x)^k$.

The following theorem establishes its pseudorandomness based on the DDH assumption.

Theorem 4.2. *$F_k(\mathsf{H}(x))$ is a family of PRF assuming H is a random oracle and the DDH assumption holds w.r.t.* $\mathsf{GroupGen}(1^\kappa) \to (\mathbb{G}, g, p)$.

Proof. We now reduce the pseudorandomness of $F_k(\mathsf{H}(\cdot))$ to the hardness of DDH problem. Let $\mathcal{B}$ be an adversary against the DDH problem. Given a DDH challenge instance $(g^a, g^{b_1}, \ldots, g^{b_n}, g^{c_1}, \ldots, g^{c_n})$, $\mathcal{B}$ interacts with an adversary $\mathcal{A}$ in the pseudorandomness experiment, with the aim to determine if $c_i = ab_i$ or random values. $\mathcal{A}$, and simulates the random oracle H and real-or-random oracle as below:

- Setup: $\mathcal{B}$ sends $pp = (\mathbb{G}, g, p, \mathsf{H})$ to $\mathcal{A}$, and implicitly sets a as the key of PRF.
- Random oracle query: for random oracle query $\langle x_i \rangle$, $\mathcal{B}$ programs $\mathsf{H}(x_i) := g^{b_i}$.

- Real-or-random query: without loss of generality, it is safe to assume adversary has already made the corresponding RO queries before making the evaluation queries. For evaluation query $\langle x_i \rangle$, $\mathcal{B}$ returns $y_i := g^{c_i}$ to $\mathcal{A}$.

- Guess: $\mathcal{A}$ makes a guess $\beta \in \{0, 1\}$, where '0' indicates real mode and '1' indicates random mode. $\mathcal{B}$ forwards β to its own challenger.

Clearly, if $c_i = ab_i$ for all $i \in [n]$, then $\mathcal{A}$ simulates the real oracle. If c_i are random values, then $\mathcal{A}$ simulates the random oracle. Thereby, $\mathcal{B}$ breaks the DDH assumption with the same advantage as $\mathcal{A}$ breaks the pseudorandomness of $F_k(\mathsf{H}(\cdot))$. □

Remark 4.2. (Weak) PRF can be build from weak pseudorandom group action (c.f. Definition in Appendix A.1) in a similar way.

5 Commutative Weak Pseudorandom Function

5.1 Definition of Commutative Weak PRF

We first formally define two standard properties for keyed functions.

Composable. For a family of keyed functions $F : K \times D \to R$, F is 2-composable if $R \subseteq D$, namely, for any $k_1, k_2 \in K$, the function $F_{k_1}(F_{k_2}(\cdot))$ is well-defined. In this work, we are interested in a special case namely $R = D$.

Commutative. For a family of composable keyed functions, we say it is commutative if:

$$\forall k_1, k_2 \in K, \forall x \in D : F_{k_1}(F_{k_2}(x)) = F_{k_2}(F_{k_1}(x))$$

It is easy to see that the standard pseudorandomness denies commutative property. Consider the following attack against the standard pseudorandomness of F_k as below: the adversary $\mathcal{A}$ picks $k' \xleftarrow{R} K$, $x \xleftarrow{R} D$, and then queries the real-or-random oracle at point $F_{k'}(x)$ and point x respectively, receiving back responses y' and y. $\mathcal{A}$ then outputs '1' iff $F_{k'}(y) = y'$. Clearly, $\mathcal{A}$ breaks the pseudorandomness with advantage $1/2 - \mathsf{negl}(\lambda)$. Provided commutative property exists, the best security we can expect is weak pseudorandomness. Looking ahead, weak pseudorandomness and commutative property may co-exist based on some well-studied assumptions.

Definition 5.1 (Commutative Weak PRF)**.** Let F be a family of keyed functions $K \times D \to D$. F is called commutative weak PRF if it satisfies weak pseudorandomness and commutative property simultaneously.

Further generalization. Instead of sticking to one family of keyed functions, commutative property can be defined over two families of keyed functions. Let F be a family of weak PRF from $K \times D$ to D, G be a family of weak PRFs $S \times D$ to D. If the following equation holds,

$$\forall k \in K, s \in S, \forall x \in D : F_k(G_s(x)) = G_s(F_k(x))$$

we say (F, G) is a tuple of commutative weak PRF.

Remark 5.1. We note that our notion of commutative weak PRF (cwPRF) is similar to but strictly weaker than a previous notion called commutative encryption [AES03]. The difference is that cwPRF neither requires F_k be a permutation nor F_k^{-1} be efficiently computable.

5.2 Construction of Commutative Weak PRF

We observe that the weak PRF construction presented in Section 4.1 already satisfies commutative property. This gives us a simple cwPRF construction from the DDH assumption.

Parameters:

- Common input: wcwPRF $F : K \times D \to D$, hash function $\mathsf{H} : \{0,1\}^\ell \to D$.
- Input of server P_1: $Y = \{y_1, \dots, y_n\} \subseteq \{0,1\}^\ell$.
- Input of client P_2: $X = \{x_1, \dots, x_n\} \subseteq \{0,1\}^\ell$ (should be interpreted as a vector).

Protocol:

1. P_1 picks $k_1 \xleftarrow{\text{R}} K$, then sends $\{F_{k_1}(\mathsf{H}(y_i))\}_{i \in [n]}$ to P_2.
2. P_2 picks $k_2 \xleftarrow{\text{R}} K$, then computes and sends $\{F_{k_2}(\mathsf{H}(x_i)))\}_{i \in [n]}$ to P_1. P_2 also computes $\{F_{k_2}(F_{k_1}(\mathsf{H}(y_i))))\}_{i \in [n]}$, picks a random permutation $\pi \xleftarrow{\text{R}} [n]$, then sends $\{F_{k_2}(F_{k_1}(\mathsf{H}(y_{\pi(i)}))\}_{i \in [n]}$ to P_1. An alternative choice instead of explicit shuffle is inserting $\{F_{k_2}(F_{k_1}(\mathsf{H}(y_i)))\}_{i \in [n]}$ to a Bloom filter, then sends the resulting filter to P_1. We slightly abuse the notation, and still use Ω to denote the Bloom filter.
3. P_1 computes $\{F_{k_1}(F_{k_2}(\mathsf{H}(x_i)))\}_{i \in [n]}$, then sets $e_i = 1$ iff $F_{k_1}(F_{k_2}(\mathsf{H}(x_i))) \in \Omega$.

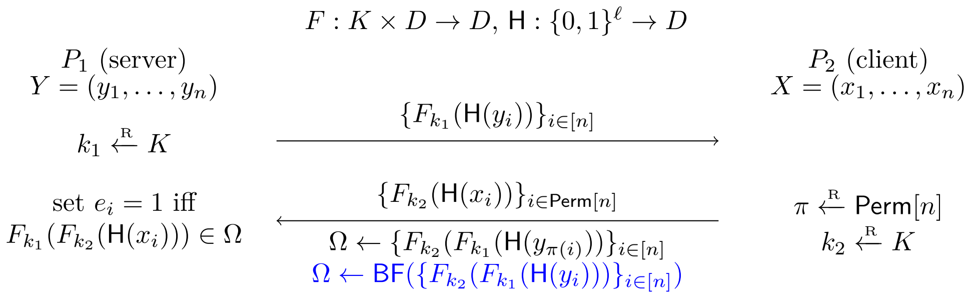

$F : K \times D \to D$, $\mathsf{H} : \{0,1\}^\ell \to D$
P_1 (server)
$Y = (y_1, \dots, y_n)$
$k_1 \xleftarrow{\text{R}} K$
$\{F_{k_1}(\mathsf{H}(y_i))\}_{i \in [n]}$
P_2 (client)
$X = (x_1, \dots, x_n)$
set $e_i = 1$ iff
$F_{k_1}(F_{k_2}(\mathsf{H}(x_i))) \in \Omega$
$\{F_{k_2}(\mathsf{H}(x_i))\}_{i \in \mathsf{Perm}[n]}$
$\Omega \leftarrow \{F_{k_2}(F_{k_1}(\mathsf{H}(y_{\pi(i)}))\}_{i \in [n]}$
$\Omega \leftarrow \mathsf{BF}(\{F_{k_2}(F_{k_1}(\mathsf{H}(y_i)))\}_{i \in [n]})$
$\pi \xleftarrow{\text{R}} \mathsf{Perm}[n]$
$k_2 \xleftarrow{\text{R}} K$

Figure 5: Multi-query RPMT from commutative weak PRF

5.3 mqRPMT from Commutative Weak PRF

In Figure 5, we show how to build mqRPMT from cwPRF $F : K \times D \rightarrow D$ and cryptographic hash function $\mathsf{H} : \{0,1\}^\ell \rightarrow D$.

Remark 5.2. We observe that thanks to the nice properties of cwPRF, the same cwPRF-based mqRPMT protocol can also be tweaked to permuted mqPMT by checking if $\hat{F}_k(\mathsf{H}(y_{\pi(i)})) \in \{\hat{F}_k(\mathsf{H}(x_i))\}$.

Correctness. The above protocol is correct except the case E that $F_{k_1}(F_{k_2}(\mathsf{H}(x))) = F_{k_1}(F_{k_2}(\mathsf{H}(y)))$ for some $x \neq y$ occurs. We further divide E to E_0 and E_1. E_0 denotes the case that $\mathsf{H}(x) = \mathsf{H}(y)$. E_1 denotes the case that $\mathsf{H}(x) \neq \mathsf{H}(y)$ but $F_{k_1}(F_{k_2}(\mathsf{H}(x))) = F_{k_1}(F_{k_2}(\mathsf{H}(y)))$, which can further be divided into sub-cases E_{10}—$F_{k_2}(\mathsf{H}(x)) = F_{k_2}(\mathsf{H}(y))$ and E_{11}—$F_{k_2}(\mathsf{H}(x)) \neq F_{k_2}(\mathsf{H}(y))$ but $F_{k_1}(F_{k_2}(\mathsf{H}(x))) = F_{k_1}(F_{k_2}(\mathsf{H}(y)))$. By the collision resistance of H and union bound, we have $\Pr[E_0] = 2^{-\kappa+2n}$. By the weak pseudorandomness of F and union bound, we have $\Pr[E_{10}] = \Pr[E_{11}] = 2^{-\ell+2n}$. Therefore, we have $\Pr[E] \leq \Pr[E_0] + \Pr[E_{10}] + \Pr[E_{11}] = 2^{-\kappa+2n} + 2^{-\ell+2n+1}$.

Theorem 5.1. *The multi-query RPMT protocol described in Figure 5 is secure in the semi-honest model assuming H is a random oracle and F is a family of cwPRFs.*

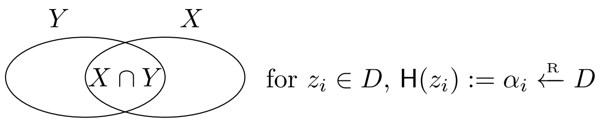
Proof. We exhibit simulators Sim_{P_1} and Sim_{P_2} for simulating corrupt P_1 and P_2 respectively, and argue the indistinguishability of the simulated transcript from the real execution. Let $|X \cap Y| = m$.

Security against corrupt client. Sim_{P_2} simulates the view of corrupt client P_2, which consists of P_2's randomness, input, output and received messages. We formally show Sim_{P_2}'s simulation is indistinguishable from the real execution via a sequence of hybrid transcripts.

Hybrid$_0$: P_2's view in the real protocol.

Hybrid$_1$: Given P_2's input X, Sim_{P_2} chooses the randomness for P_1 (i.e., picks $k_1 \xleftarrow{\text{R}} K$), and simulates with the knowledge of Y.

- RO query: Sim_{P_2} emulates the random oracle H honestly. For each query $\langle z_i \rangle$, Sim_{P_2} picks $\alpha_i \xleftarrow{\text{R}} D$, and assigns $\mathsf{H}(z_i) := \alpha_i$.
- Sim_{P_2} outputs $(F_{k_1}(\mathsf{H}(y_1)), \ldots, F_{k_1}(\mathsf{H}(y_n)))$.


Y
X
$X \cap Y$
for $z_i \in D$, $\mathsf{H}(z_i) := \alpha_i \xleftarrow{\text{R}} D$

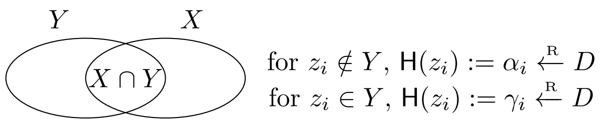
Clearly, Sim_{P_2}'s simulated view in Hybrid$_1$ is identical to P_2's real view.

Hybrid$_2$: Sim_{P_2} does not choose the randomness for P_1 (i.e., picks $k_1 \xleftarrow{\text{R}} K$), and simulates without the knowledge of Y. It emulates the random oracle H honestly as before, and only changes the simulation of P_1's message.

- Sim_{P_2} outputs $(\eta_1, \ldots, \eta_n)$ where $\eta_i \xleftarrow{\text{R}} D$.

We argue that the simulated view in Hybrid$_1$ and Hybrid$_2$ are computationally indistinguishable. More precisely, a PPT adversary $\mathcal{A}$ (with knowledge of X and Y) against cwPRF (with secret key k) is given n tuples (γ_i, η_i) where $\gamma_i \xleftarrow{\text{R}} D$, and is asked to distinguish if $\eta_i = F_k(\gamma_i)$ or η_i are random values. $\mathcal{A}$ implicitly sets P_1's randomness $k_1 := k$, and simulates as below.

- RO query: for each random oracle query $\langle z_i \rangle$, if $z_i \notin Y$, picks $\alpha_i \xleftarrow{\text{R}} D$ and sets $\mathsf{H}(z_i) := \alpha_i$; if $z_i \in Y$, sets $\mathsf{H}(z_i) := \gamma_i$.
- outputs $(\eta_1, \ldots, \eta_n)$.


Y
X
$X \cap Y$
for $z_i \notin Y$, $\mathsf{H}(z_i) := \alpha_i \xleftarrow{\text{R}} D$
for $z_i \in Y$, $\mathsf{H}(z_i) := \gamma_i \xleftarrow{\text{R}} D$

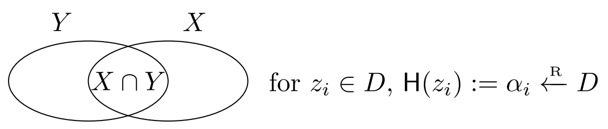
If $\eta_i = F_k(\gamma_i)$ for $i \in [n]$, then $\mathcal{A}$'s simulation is identical to Hybrid$_1$. If η_i are random values, then $\mathcal{A}$'s simulation is identical to Hybrid$_2$.

Security against corrupt server. Sim_{P_1} simulates the view of corrupt server P_1, which consists of P_1's randomness, input, output and received messages. We formally show Sim_{P_1}'s simulation is indistinguishable from the real execution via a sequence of hybrid transcripts.

Hybrid$_0$: P_1's view in the real protocol.

Hybrid$_1$: Given P_1's input Y and output $(e_1, \ldots, e_n)$, Sim_{P_1} chooses the randomness for P_2 (i.e., picks $k_2 \xleftarrow{\mathrm{R}} K$ and a random permutation π over $[n]$), and simulates with the knowledge of X.

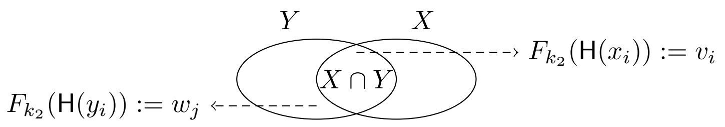
- RO queries: Sim_{P_1} emulates the random oracle H honestly. For each query $\langle z_i \rangle$, Sim_{P_1} picks $\alpha_i \xleftarrow{\mathrm{R}} D$ and assigns $\mathsf{H}(z_i) := \alpha_i$.
- Sim_{P_1} outputs $\{F_{k_2}(\mathsf{H}(x_i))\}_{i\in[n]}$ and $\Omega \leftarrow \{F_{k_2}(F_{k_1}(\mathsf{H}(y_{\pi(i)}))\}_{i\in[n]}$.


Y
X
$X \cap Y$
for $z_i \in D$, $\mathsf{H}(z_i) := \alpha_i \xleftarrow{\mathrm{R}} D$

Clearly, Sim_{P_1}'s simulation in Hybrid$_1$ is identical to the real view of P_1.

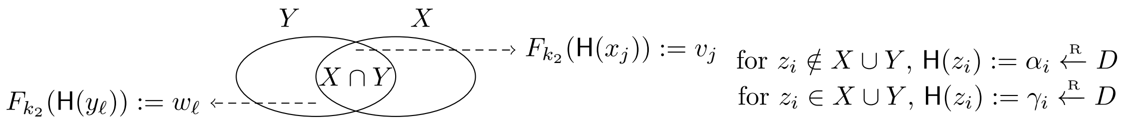
Hybrid$_2$: Sim_{P_1} does not choose randomness for P_1, and simulates without the knowledge of X. It simulates the random oracle H honestly as before, and changes its simulation of P_2's message. Let m be the Hamming weight of $(e_1, \ldots, e_n)$.

- Sim_{P_1} picks $v_i \xleftarrow{\mathrm{R}} D$ for $i \in [n]$ (associated with $F_{k_2}(\mathsf{H}(x_i))$ where $x_i \in X$), outputs $\{v_i\}_{i\in[n]}$; picks $w_j \xleftarrow{\mathrm{R}} D$ for $j \in [n-m]$ (associated with $F_{k_2}(\mathsf{H}(y_j))$ where $y_j \in Y - X \cap Y$), outputs a random permutation of $(\{F_{k_1}(v_i)\}_{e_i=1}, \{F_{k_1}(w_j)\}_{j\in[n-m]})$.


Y
X
$X \cap Y$
$F_{k_2}(\mathsf{H}(x_i)) := v_i$
$F_{k_2}(\mathsf{H}(y_i)) := w_j$

We argue that the view in Hybrid$_1$ and Hybrid$_2$ are computationally indistinguishable. More precisely, a PPT adversary $\mathcal{A}$ (with knowledge of X and Y) against cwPRF are given $2n - m$ tuples (γ_i, η_i) where $\gamma_i \xleftarrow{\mathrm{R}} D$, and is asked to determine if $\eta_i = F_k(\gamma_i)$ or random values. $\mathcal{A}$ implicitly sets P_2's randomness $k_2 := k$, picks $k_1 \xleftarrow{\mathrm{R}} K$.

- RO queries: for $z_i \notin X \cup Y$, picks $\alpha_i \xleftarrow{\mathrm{R}} D$ and assigns $\mathsf{H}(z_i) := \alpha_i$; for $z_i \in X \cup Y$, assigns $\mathsf{H}(z_i) := \gamma_i$.
- For each $z_i \in X$, $\mathcal{A}$ picks out the associated η_i to form $\{v_j\}_{j\in[n]}$; for each $z_i \in Y - X \cap Y$, $\mathcal{A}$ picks out the associated η_i to form $\{w_\ell\}_{\ell\in[n-m]}$. Finally, $\mathcal{A}$ outputs $\{v_j\}_{j\in[n]}$ and a random permutation of $(\{F_{k_1}(v_j)\}_{x_j \in X\cap Y}, \{F_{k_1}(w_\ell)\}_{\ell\in[n-m]})$.


Y
X
$X \cap Y$
$F_{k_2}(\mathsf{H}(x_j)) := v_j$
$F_{k_2}(\mathsf{H}(y_\ell)) := w_\ell$
for $z_i \notin X \cup Y$, $\mathsf{H}(z_i) := \alpha_i \xleftarrow{\mathrm{R}} D$
for $z_i \in X \cup Y$, $\mathsf{H}(z_i) := \gamma_i \xleftarrow{\mathrm{R}} D$

If $\eta_i = F_k(\gamma_i)$, then $\mathcal{A}$'s simulation is identical to Hybrid$_1$. If η_i are random values, then $\mathcal{A}$'s simulation is identical to Hybrid$_2$.

This proves the theorem. $\square$

6 Permuted Oblivious Pseudorandom Function

6.1 Definition of Permuted OPRF

An oblivious pseudorandom function (OPRF) [FIPR05] is a two-party protocol in which the sender learns a PRF key k and a receiver learns $F_k(x_1), \dots, F_k(x_n)$, where F is a pseudorandom function (PRF) and $(x_1, \dots, x_n)$ are the receiver's inputs. Nothing about the receiver's inputs is revealed to the sender and nothing more about the key k is revealed to the receiver.

We consider an extension of OPRF which we called permuted OPRF. Roughly speaking, the sender additionally picks a random permutation π over $[n]$, and the receiver learns its PRF values in permuted order, namely, $y_i = F_k(x_{\pi(i)})$. In Figure 6 we formally define the ideal functionality for pOPRF.

Parameters: number of OPRF queries n.

Functionality: On input set $X = (x_1, \dots, x_n) \subseteq \{0,1\}^\ell$ from the receiver:

1. Choose a random PRF key k and a random permutation π over $[n]$.
2. Give k and π to the sender and $y_i = F_k(x_{\pi(i)})$ to the receiver.

Figure 6: Ideal functionality $\mathcal{F}_{\mathsf{pOPRF}}$ for permuted OPRF

6.2 Construction of Permuted OPRF

As we sketched in the introduction part, we can create a permuted OPRF from enhanced cwPRF (F_k, G_s), in which G_s is a weak permutation. At a high level, the unified masking procedure is done by applying a weak PRF $F_r(\cdot)$ to $\mathsf{H}(x)$, and the unmasking process is enabled by the commutative property of (F_k, G_s) and the fact that $G_s(\cdot)$ is an efficiently invertible permutation. We depict the construction as below.

$$\begin{array}{ccc}
 & F : K \times D \to D,\ G : S \times D \to D & \\
 & \mathsf{H} : \{0,1\}^\ell \to D & \\
P_1 \text{ (sender)} & & P_2 \text{ (receiver)} \\
 & & X = (x_1, \dots, x_n) \\
 & \xleftarrow{\quad \{G_s(\mathsf{H}(x_i))\}_{i \in [n]} \quad} & s \xleftarrow{\mathrm{R}} S \\
k \xleftarrow{\mathrm{R}} K,\ \pi \xleftarrow{\mathrm{R}} [n] & \xrightarrow{\quad \{F_k(G_s(\mathsf{H}(x_{\pi(i)}))\}_{i \in [n]} \quad} & F_k(\mathsf{H}(x_{\pi(i)})) \leftarrow G_s^{-1}(F_k(G_s(\mathsf{H}(x_{\pi(i)}))))
\end{array}$$

Figure 7: Permuted OPRF from cwPRF

Theorem 6.1. *The above permuted OPRF protocol described in Figure 7 is secure in the semi-honest model assuming* H *is a random oracle,* (F_k, G_s) *is a tuple of cwPRF and* G_s *is a weak permutation.*

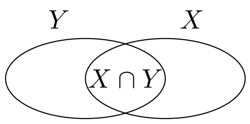
Proof. We exhibit simulators Sim_{P_1} and Sim_{P_2} for simulating corrupt P_1 and P_2 respectively, and argue the indistinguishability of produced transcript from the real execution.

Security against corrupt sender. Sim_{P_1} simulates the view of corrupt sender P_1, which consists of P_1's randomness, input, output and received messages. We formally show Sim_{P_1}'s simulation is indistinguishable from the real execution via a sequence of hybrid transcripts.

Hybrid$_0$: P_1's view in the real protocol.

Hybrid$_1$: Given P_1's output k and π, Sim_{P_1} chooses the randomness s for P_2, and simulates with the knowledge of $X = (x_1, \dots, x_n)$:

- RO queries: Sim_{P_1} honestly emulates random oracle H. For every query $\langle z_i \rangle$, picks $\alpha_i \xleftarrow{R} D$ and assigns $\mathsf{H}(z_i) := \alpha_i$.
- Sim_{P_1} outputs $(G_s(\beta_1), \ldots, G_s(\beta_n))$, where $\mathsf{H}(x_i) = \beta_i$.



for $z_i \in D$, $\mathsf{H}(z_i) := \alpha_i \xleftarrow{R} D$

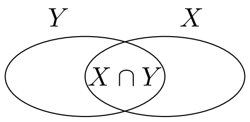
Clearly, Sim_{P_1}'s simulated view in Hybrid$_1$ is identical to P_1's real view.

Hybrid$_2$: Sim_{P_1} does not choose the randomness for P_2, and simulates without the knowledge of X. It honestly emulates random oracle H as in Hybrid$_1$, and only changes the simulation of P_2's message.

- Sim_{P_1} outputs $(\eta_1, \ldots, \eta_n)$ where $\eta_i \xleftarrow{R} D$.

We argue that the view in Hybrid$_1$ and Hybrid$_2$ are computationally indistinguishable. Let $\mathcal{A}$ be a PPT adversary against the weak pseudorandom of G_s. Given a real-or-random oracle $\mathcal{O}_{\mathsf{ror}}(\cdot)$, $\mathcal{A}$ is asked to distinguish which mode he is in. $\mathcal{A}$ queries the $\mathcal{O}_{\mathsf{ror}}(\cdot)$ n times, and obtains (γ_i, η_i) in return. $\mathcal{A}$ then simulates (with the knowledge of X) as below:

- RO queries: for each query $\langle z_i \rangle$, if $z_i \notin X$, picks $\alpha_i \xleftarrow{R} D$ and assigns $\mathsf{H}(z_i) := \alpha_i$; if $z_i \in X$, assigns $\mathsf{H}(x_i) := \gamma_i$.
- Outputs $(\eta_1, \ldots, \eta_n)$.



for $z_i \notin X$, $\mathsf{H}(z_i) := \alpha_i \xleftarrow{R} D$
for $z_i \in X$, $\mathsf{H}(z_i) := \gamma_i$

Clearly, if $\eta_i = G_s(\gamma_i)$, $\mathcal{A}$ simulates Hybrid$_1$. Else, it simulates Hybrid$_2$. Thereby, Sim_{P_1}'s simulated view is computationally indistinguishable to P_1's real view.

Security against corrupt receiver. Sim_{P_2} simulates the view of corrupt receiver P_2, which consists of P_2's randomness, input, output and received messages. We formally show Sim_{P_2}'s simulation is indistinguishable from the real execution via a sequence of hybrid transcripts.

Hybrid$_0$: P_2's view in the real protocol.

Hybrid$_1$: Given P_2's input $X = (x_1, \ldots, x_n)$ and output $\{F_k(\mathsf{H}(x_{\pi(i)}))\}_{i \in [n]}$, Sim_{P_2} emulates the random oracle H honestly, picks $s \xleftarrow{R} \mathbb{Z}_p$, simulates message from P_1 as $\{G_s(F_k(\mathsf{H}(x_{\pi(i)})))\}_{i \in [n]}$.

According to the commutative property of cwPRF, Sim_{P_2}'s simulated view is identical to the real view.

This proves the theorem. □

Observe that the cwPRF construction presented in Section 5.2 already satisfies the enhanced property that G_s being a permutation. Plugging it to the above generic construction, we obtain a concrete pOPRF protocol as described in Figure 8.

The security of the above pOPRF protocol is guaranteed by Theorem 6.1 and the security of the underlying cwPRF, which is in turn based on the DDH assumption. For completeness, we provide a direct security proof based on the DDH assumption in Appendix C.1.

6.3 mqRPMT from Permuted OPRF

In Figure 9, we show how to build mqRPMT from permuted OPRF for $F : K \times D \to R$. For simplicity, we assume that $\{0,1\}^\ell \subseteq D$. Otherwise, we can always map $\{0,1\}^\ell$ to D via collision resistant hash function.

Parameters:

- Common input: $F : \mathbb{Z}_p \times \mathbb{G} \to \mathbb{G}$, hash function $\mathsf{H} : \{0,1\}^\ell \to \mathbb{G}$.
- Input of receiver P_2: $X = \{x_1, \ldots, x_n\} \subseteq \{0,1\}^\ell$.

Protocol:

1. P_2 picks $s \xleftarrow{\text{R}} \mathbb{Z}_p$, then sends $(\mathsf{H}(x_1)^s, \ldots, \mathsf{H}(x_n)^s)$ to the sender P_1.
2. P_1 picks $k \xleftarrow{\text{R}} \mathbb{Z}_p$ and computes $(\mathsf{H}(x_1)^{sk}, \ldots, \mathsf{H}(x_n)^{sk})$, then picks a random permutation π over $[n]$ and sends $y'_i = \mathsf{H}(x_{\pi(i)})^{sk}$ for $i \in [n]$ to P_2.
3. P_1 outputs k and π.
4. P_2 outputs $y_i = (y'_i)^{s^{-1}}$ for each $i \in [n]$.

Figure 8: Permuted OPRF from the DDH assumption

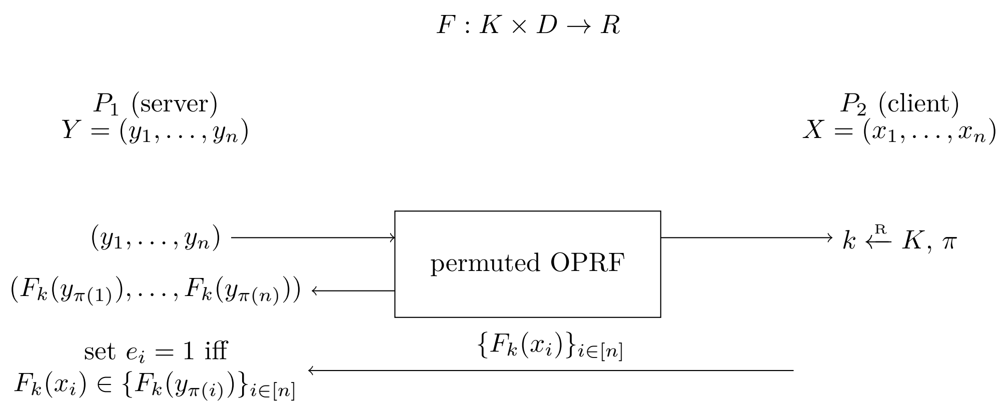
Parameters:

- Common input: permuted OPRF for $F : K \times D \to R$.
- Input of server P_1: $Y = \{y_1, \ldots, y_n\} \subseteq \{0,1\}^\ell$.
- Input of client P_2: $X = \{x_1, \ldots, x_n\} \subseteq \{0,1\}^\ell$.

Protocol:

1. P_1 with inputs $Y = \{y_1, \ldots, y_n\}$ and P_2 invoke the permuted OPRF protocol. At the end of the protocol, P_1 obtains $\{F_k(y_{\pi(i)})\}_{i\in[n]}$, P_2 obtains k and a permutation π over $[n]$.
2. P_2 computes and sends $(F_k(x_1), \ldots, F_k(x_n)))$ to P_1.
3. P_1 sets $e_i = 1$ iff $F_k(x_i) \in \{F_k(y_{\pi(i)})\}_{i\in[n]}$.

Figure 9: mqRPMT from permuted OPRF



Correctness. The above protocol is correct except the case $E = \vee_{i,j} E_{ij}$ oucurs, where E_{ij} denotes $F_k(x_i) = F_k(y_j)$ but $x_i \neq y_j$. By pseudorandomness of F, we have $\Pr[E_{ij}] = 2^{-\ell}$. Apply the union bound, we have $\Pr[E] \leq n^2 \cdot \Pr[E_{ij}] \leq n^2/2^\ell = \mathsf{negl}(\lambda)$.

Theorem 6.2. *The above mqRPMT protocol described in Figure 9 is secure in the semi-honest model assuming the security of permuted OPRF F.*

Proof. We exhibit simulators Sim_{P_1} and Sim_{P_2} for simulating corrupt P_1 and P_2 respectively, and argue the indistinguishability of the produced transcript from the real execution. Let $|X \cap Y| = m$.

Security against corrupt client. Sim_{P_2} simulates the view of corrupt client P_2, which consists of P_2's randomness, input, output and received messages. We formally show Sim_{P_2}'s simulation is indistinguishable from the real execution via a sequence of hybrid transcripts.

Hybrid$_0$: P_2's view in the real protocol.

Hybrid$_1$: Sim_{P_2} simply picks k and π, then invokes the simulator for P_2 in the permuted OPRF with (k, π) as output. By the semi-honest security of permuted OPRF on P_2's side, the simulation is indistinguishable to the real view.

Security against corrupt server. Sim_{P_1} simulates the view of corrupt server P_1, which consists of P_1's randomness, input, output and received messages. We formally show Sim_{P_1}'s simulation is indistinguishable from the real execution via a sequence of hybrid transcripts.

Hybrid$_0$: P_1's view in the real protocol. Note that P_1's view consists of its view in stage 1 (the permuted OPRF part) and its view in stage 2.

Hybrid$_1$: Given P_1's input $Y = (y_1, \dots, y_n)$ and output $(e_1, \dots, e_n)$, Sim_{P_1} creates the simulated view as below:

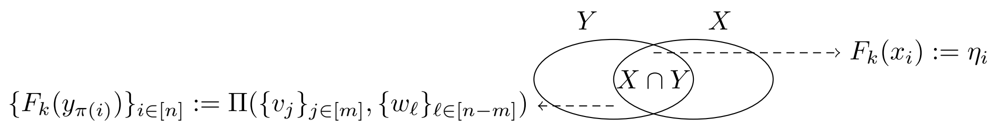
- pick a random PRF key k and a random permutation π;
- compute $(F_k(y_{\pi(1)}), \dots, F_k(y_{\pi(n)}))$, then generate its stage 1's view by invoking the simulator for P_1 of permuted OPRF with input $(y_1, \dots, y_n)$ and output $(F_k(y_{\pi(1)}), \dots, F_k(y_{\pi(n)}))$;
- generate stage 2's view $(F_k(x_1), \dots, F_k(x_n))$ using k with the knowledge of P_2's input X.

The simulated stage 2'view is identical to that in the real one. By the semi-honest security of permuted OPRF on P_1' side, the stage 1's simulated view of stage 1 is computationally indistinguishable to that in the real one. Thereby, the simulated view in Hybrid$_1$ is computationally indistinguishable to the real one.

Hybrid$_2$: Sim_{P_1} creates the simulated view without the knowledge of X, and it neither picks k nor explicitly picks π:

- generate stage 2's view by outputting $(\eta_1, \dots, \eta_n)$, where $\eta_i \xleftarrow{R} R$; this implicitly sets $F_k(x_i) := \eta_i$.

- for each $e_i = 1$, pick out the associated η_i to form $\{v_j\}_{j\in[m]}$; for each $e_i = 0$, pick random values to form $\{w_\ell\}_{\ell\in[n-m]}$; pick a random permutation Π of $(\{v_j\}_{j\in[m]}, \{w_\ell\}_{\ell\in[n-m]})$, treat the result as $(F_k(y_{\pi(1)}), \ldots, F_k(y_{\pi(n)}))$ (note that the real permutation π is unknown to the simulator since it does not know $X \cap Y$); then generate its stage 1's view by invoking the simulator for P_1 of permuted OPRF with input $(y_1, \ldots, y_n)$ and output $(F_k(y_{\pi(1)}), \ldots, F_k(y_{\pi(n)}))$.


Y
X
$F_k(x_i) := \eta_i$
$X \cap Y$
$\{F_k(y_{\pi(i)})\}_{i\in[n]} := \Pi(\{v_j\}_{j\in[m]}, \{w_\ell\}_{\ell\in[n-m]})$

We argue that the simulated views in Hybrid$_1$ and Hybrid$_2$ are computationally indistinguishable based on the pseudorandomness of F. Let $\mathcal{A}$ be an adversary against F. Given X and Y, $\mathcal{A}$ simulates as below:

- query the real-or-random oracle $\mathcal{O}_{\mathsf{ror}}(\cdot)$ with $(x_1, \ldots, x_n)$ and obtain $(\eta_1, \ldots, \eta_n)$, output $(\eta_1, \ldots, \eta_n)$.
- pick a random permutation π;
- query the real-or-random oracle with $(y_{\pi(1)}, \ldots, y_{\pi(n)})$ and obtain $(\zeta_1, \ldots, \zeta_n)$ in return; then generate its stage 1's view by invoking the simulator for P_1 of permuted OPRF with input $(y_1, \ldots, y_n)$ and output $(\zeta_1, \ldots, \zeta_n)$.

Clearly, if $\mathcal{A}$ queries the real oracle, then its simulation is identical to that Hybrid$_1$. Else, its simulation is identical to that Hybrid$_2$. This reduces the computational indistinguishability of views in Hybrid$_1$ and Hybrid$_2$ to the pseudorandomness of $F_k(\cdot)$. Therefore, Sim_{P_1}'s simulation is indistinguishable to the real one.

This proves the theorem. $\square$

7 mqRPMT from Sigma-mqPMT

7.1 Sigma-mqPMT

Private membership test (PMT) protocol [PSZ14] is a two-party protocol in which the client with input x learns whether or not its item is in the input set Y of the server. PMT can be viewed as a special case of private keyword search protocol [FIPR05] by setting the payload as any indication string. We consider three-move PMT, which we refer to Sigma-PMT hereafter.

Sigma-PMT proceeds via following pattern.

1. Server P_1 sends the first round message a to client P_2, which is best interpreted as an encoding of Y.
2. Client P_2 sends query q w.r.t. to his item x.
3. Server P_1 responds with z.

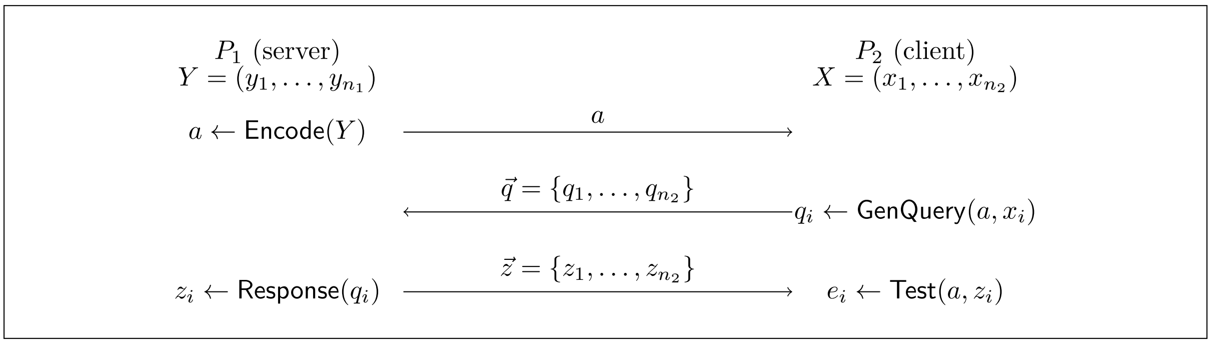
After receiving z, client P_2 can decide if $x \in Y$ by running $\mathsf{Test}(a, x, q, z)$. The basic notion of Sigma PMT allows the client P_2 to test for a single item. While this procedure can be repeated several times, one may seek for more efficient protocol allowing the client to test n items at reduced communication cost and round complexity. To this end, we introduce the following two properties for Sigma-PMT:

- **Reusable:** The first round message is performed by the server P_1 once and for all.
- **Context-independent:** Each test query q_i is only related to the element x_i under test and the randomness of P_2.

The first property helps to reduce communication cost, while the second property admits parallelization, hence the round complexity is unchanged even when handling multiple items. Sigma-PMT may enjoy an additional property:

- **Stateless:** For any x_i and associated (q_i, z_i), $\mathsf{Test}(a, x_i, q_i, z_i)$ can work in a memoryless way, namely, without looking at (x_i, q_i). In this case, the test algorithm can be simplified as $\mathsf{Test}(a, z_i)$.

By running Sigma-PMT with reusable, context-independent, and stateless properties in parallel, we obtain mqPMT with three-move pattern (depicted in Figure 10), which we refer to as Sigma-mqPMT.



P_1 (server)
$Y = (y_1, \ldots, y_{n_1})$
P_2 (client)
$X = (x_1, \ldots, x_{n_2})$
$a \leftarrow \mathsf{Encode}(Y)$
a
$\vec{q} = \{q_1, \ldots, q_{n_2}\}$
$q_i \leftarrow \mathsf{GenQuery}(a, x_i)$
$\vec{z} = \{z_1, \ldots, z_{n_2}\}$
$z_i \leftarrow \mathsf{Response}(q_i)$
$e_i \leftarrow \mathsf{Test}(a, z_i)$

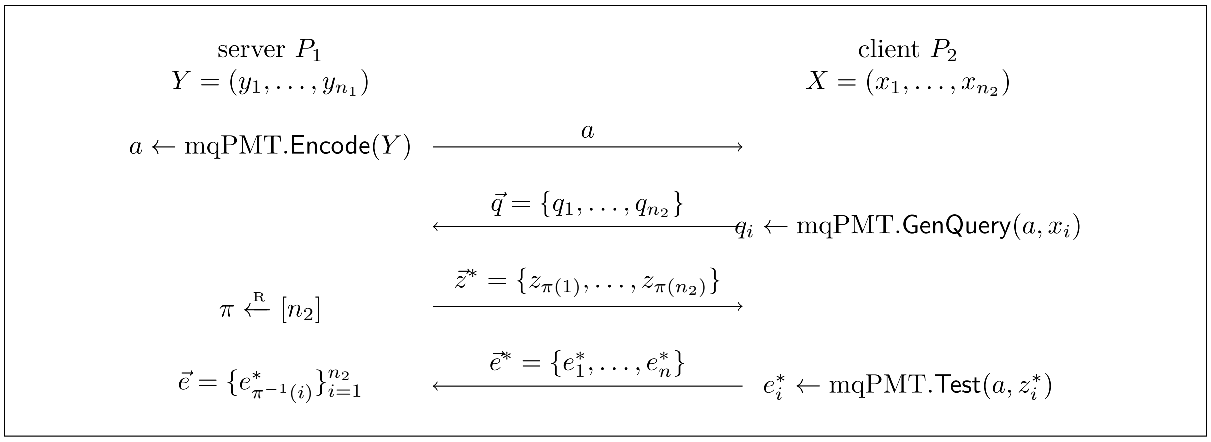
Figure 10: Sigma-mqPMT

To reduce the semi-honest security of mqRPMT* to that of Sigma-mqPMT, we assume the simulator $\mathsf{Sim}(X, \vec{e})$ for client P_2 is composed of two sub-routines $(\mathsf{Sim}', \mathsf{Sim}'')$, and satisfies the following properties:

- **Locality:** $z_i \approx \mathsf{Sim}'(e_i; r_i)$, a.k.a. the i-th response can be emulated via invoking a sub-routine $\mathsf{Sim}'(e_i)$ with independent random coins r_i;
- **Order invariance:** $a \approx \mathsf{Sim}''(\{e_{\pi(i)}, r_{\pi(i)}\}_{i \in [n_2]}; s)$, where π could be an arbitrary permutation over $[n_2]$, s is the random coins.

7.2 Connection to Sigma-mqPMT

Next, we show a generic construction of mqRPMT* from Sigma-mqPMT. With the nice properties of Sigma-mqPMT, the construction is pretty simple, a.k.a. having the server P_1 shuffle the last move message in Sigma-mqPMT (yielding permuted mqPMT upon this step), then having the client P_2 sends the test results back to P_1, and finally P_1 recovers the indication bits in the right order. We formally describe the construction in Figure 11.



server P_1
$Y = (y_1, \ldots, y_{n_1})$
client P_2
$X = (x_1, \ldots, x_{n_2})$
$a \leftarrow \text{mqPMT}.\mathsf{Encode}(Y)$
a
$\vec{q} = \{q_1, \ldots, q_{n_2}\}$
$q_i \leftarrow \text{mqPMT}.\mathsf{GenQuery}(a, x_i)$
$\pi \xleftarrow{\text{R}} [n_2]$
$\vec{z}^* = \{z_{\pi(1)}, \ldots, z_{\pi(n_2)}\}$
$\vec{e} = \{e^*_{\pi^{-1}(i)}\}_{i=1}^{n_2}$
$\vec{e}^* = \{e^*_1, \ldots, e^*_n\}$
$e^*_i \leftarrow \text{mqPMT}.\mathsf{Test}(a, z^*_i)$

Figure 11: mqRPMT* from Sigma-mqPMT

Theorem 7.1. *The above mqRPMT* protocol depicted in Figure 11 is secure in the semi-honest model assuming the semi-honest security of the starting Sigma-mqPMT protocol.*

Proof. We exhibit simulators Sim_{P_1} and Sim_{P_2} for simulating corrupt server P_1 and corrupt client P_2 respectively. Let $|X \cap Y| = m$.

Security against corrupt client. Sim_{P_2} simulates the view of corrupt client P_2, which consists of P_2's randomness, input, output and received messages.

We argue that the output of Sim_{P_2} is indistinguishable from the real execution. We formally show Sim_{P_2}'s simulation is indistinguishable from the real execution via a sequence of hybrid transcripts.

Hybrid$_0$: P_2's view in the real protocol.

Hybrid$_1$: Sim_{P_2} chooses the randomness for P_1, and simulates with the knowledge of Y. Clearly, Sim_{P_2}'s simulation is identical to the real view of P_2.

Hybrid$_2$: Sim_{P_2} does not choose the randomness for P_1, and simulates without the knowledge of Y. Instead, it invokes the Sigma-mqPMT's simulator for P_2 on his private input X and output $\vec{e}^*$ to emulate the view $(a, \vec{z}^*)$ in the following manner:

- for $1 \le i \le n_2$, run $\mathsf{Sim}'(e_i^*; r_i) \to z_i^*$, obtaining $\vec{z}^* = (z_1^*, \dots, z_n^*)$.
- run $\mathsf{Sim}''(\{(e_i^*, r_i)\}_{i \in [n_2]}; s) \to a$.

By the *locality* and *order invariance* properties, the simulated view in Hybrid$_2$ and Hybrid$_1$ are computationally indistinguishable based on semi-honest security of mqPMT on P_2 side.

Security against corrupt server. Sim_{P_1} simulates the view of corrupt server P_1, which consists of P_1's randomness, input, output and received messages. We formally show Sim_{P_2}'s simulation is indistinguishable from the real execution via a sequence of hybrid transcripts.

Hybrid$_0$: P_1's view in the real protocol.

Hybrid$_1$: Sim_{P_1} chooses the randomness for P_2, and simulates with the knowledge of X. Clearly, Sim_{P_1}'s simulation is identical to the real view of P_1.

Hybrid$_2$: Sim_{P_1} does not choose the randomness for P_2, and simulates without the knowledge of X. Instead, it first invokes the Sigma-mqPMT's simulator for P_1 on input Y to generate $\vec{q}$, then picks a random permutation π and a random $\vec{e}$ with Hamming weight m, computes $\vec{e}^* = \pi^{-1}(\vec{e})$, outputs $(\vec{q}, \vec{e}^*)$.

Clearly, the view in Hybrid$_1$ and Hybrid$_2$ are computationally indistinguishable based on the semi-honest security of Sigma-mqPMT on P_1's side.

This proves the theorem. □

Remark 7.1. As a byproduct, we note that if P_1 only permutes and sends the last move message in Sigma-mqPMT, then we obtain a standard PSI-card protocol. In this perspective, it is fair to say Sigma-mqPMT distills sufficient characteristics of what kind of PSI protocols can be converted to PSI-card with no extra overhead.

8 Applications of mqRPMT

We show how to build a PSO framework central around mqRPMT in Figure 12. We would like to briefly compare our PSI-card-sum protocol with closely related protocols [IKN+20, GMR+21] as below. As mentioned in the introduction part, the PSI-card-sum protocol is built from concrete primitives (e.g. DH-protocol, ROT-protocol, Phasing+OPPRF etc.) with general 2PC techniques or AHE schemes. This renders their protocols less general and efficient. The protocol presented in [GMR+21] is built from permuted characteristic (permuted mqRPMT under our terminology) and secret sharing. Our protocol is similar to their protocol but with the following differences. First, the mqRPMT subprotocol underlying our protocol is conceptually simpler than its permuted version. mqRPMT admits instantiations with optimal linear complexity, while the current best instantiation of permuted mqRPMT requires super-linear complexity. Second, as we pointed out in the introduction part, the protocol due to [GMR+21] deviates from the standard functionality of PSI-card-sum. In contrast, our protocol meets the standard functionality of PSI-card-sum as defined in [IKN+20]. We doing so by simply removing the constraint $\sum_{i=1}^{n} r_i = 0$ on the receiver side (as did in [GMR+21]), and having the sender sends back the masked sum value to the receiver, and the receiver finally recovers the intersection sum by unmasking.

Parameters:

- Input of receiver P_1: $Y = \{y_1, \ldots, y_{n_1}\} \subseteq \{0,1\}^\ell$.
- Input of sender P_2: $X = \{x_1, \ldots, x_{n_2}\} \subseteq \{0,1\}^\ell$ and $V = \{v_1, \ldots, v_{n_2}\}$ where $v_i \in [0, \mathrm{MAX}]$. Let p be a big integer greater than $n_2 \cdot \mathrm{MAX}$.

Protocol:

0. P_2 shuffles the set $(x_1, \ldots, x_{n_2})$ and $(v_1, \ldots, v_{n_2})$ according to the same random permutation over $[n_2]$. For simplicity, we still use the original notation to denote the vector after permutation.
1. P_1 (playing the role of server) with Y and P_2 (playing the role of client) with $X = \{x_1, \ldots, x_{n_2}\}$ invoke $\mathcal{F}_{\mathsf{mqRPMT}}$. P_1 obtains an indication bit vector $\vec{e} = (e_1, \ldots, e_{n_2})$. P_2 obtains nothing.
   - **cardinality:** P_1 learns the cardinality by calculating the Hamming weight of $\vec{e}$.
2. P_1 and P_2 invoke n_2 instances of OT via $\mathcal{F}_{\mathsf{OT}}$. P_1 uses $\vec{e}$ as the choice bits.
   - **intersection:** P_2 uses $(\perp, x_i)$ as input to the ith OT. P_1 learns $\{x_i \mid e_i = 1\}_{i \in [n_2]} = X \cap Y$.
   - **union:** P_2 uses $(x_i, \perp)$ as input to the ith OT. P_1 learns $\{x_i \mid e_i = 0\}_{i \in [n_2]} = X/Y$, and outputs $\{X/Y\} \cup Y = X \cup Y$.
   - **card-sum:** P_2 randomly picks $r_i \in \mathbb{Z}_p$ and computes $r' = \sum_{i=1}^{n_2} r_i \bmod p$. Subsequently, P_1 holding e_i and P_2 holding $(r_i, r_i + v_i)$ interact 1-out-of 2 OT n times. P_1 learns $S' = \{\sum_{i=1}^{n_2} v_i \mid e_i = 1\}_{i \in [n_2]}$, then sends S' and the Hamming weight of $\vec{e}$ to P_2. P_2 computes $S = (S' - r') \bmod p$.

Figure 12: PSO from mqRPMT

We prove the security of the above PSO framework by the case of PSU. The security proof of other functionality is similar.

Theorem 8.1. *The PSU derived from the above framework described in Figure 12 is semi-honest secure by assuming the semi-honest security of mqRPMT and OT.*

Proof. We exhibit simulators Sim_{P_1} and Sim_{P_2} for simulating corrupt P_1 and P_2 respectively, and argue the indistinguishability of the produced transcript from the real execution. Let $|X \cap Y| = m$.

Security against corrupt sender. Sim_{P_2} simulates the view of corrupt sender P_2, which consists of P_2's randomness, input, output and received messages. We formally show Sim_{P_2}'s simulation is indistinguishable from the real execution via a sequence of hybrid transcripts.

Hybrid_0: P_2's view in the real protocol. Note that P_2's view consists of two parts, i.e., the mqRPMT part of view (stage 1) and the OT part of view (stage 2).

Hybrid_1: Sim_{P_2} first invokes the simulator for client in the mqRPMT with X as input to generate the stage 1's part of view, then invokes the simulator for sender in the OT with $\{(x_i, \perp)\}_{i\in[n_2]}$ as input to generate stage 2's part of view. By the semi-honest security of mqRPMT on client side and the semi-honest security for OT on sender side, the simulation is indistinguishable to the real view via standard hybrid argument.

Security against corrupt receiver. Sim_{P_1} simulates the view of corrupt receiver P_1, which consists of P_1's randomness, input, output and received messages. We formally show Sim_{P_1}'s simulation is indistinguishable from the real execution via a sequence of hybrid transcripts.

Hybrid_0: P_1's view in the real protocol. Note that P_1's view also consists of two parts, i.e., the mqRPMT part of view (stage 1) of and the OT part of view (stage 2).

Hybrid_1: Given P_1's input $Y = (y_1, \dots, y_{n_1})$ and output $X \cup Y$, Sim_{P_1} creates the simulated view as below:

- pick a random indication vector $\vec{e} = (e_1, \dots, e_{n_2})$ with Hamming weight $m = |X \cap Y|$, then generate the output vector $\vec{z} = (z_1, \dots, z_{n_2})$ from $\vec{e}$ and $X \cup Y$ in the following manner: randomly shuffle the $(n_2 - m)$ elements in $X \backslash Y$, and assign them to z_i if $e_i = 0$, then assign $z_i = \perp$ iff $e_i = 1$; then invoke the simulator for OT receiver with input $\vec{e}$ and output $\vec{z}$ and to generate stage 2's view.
- invoke the simulator for mqRPMT server with input Y and output $\vec{e} = (e_1, \dots, e_{n_2})$ to generate stage 1's view.

It is easy to check that the distribution of $\vec{e}$ and $\vec{z}$ is identical to that (induced by the distribution of mqRPMT's input vector $(x_1, \dots, x_{n_2})$) in the real protocol. By the semi-honest security of mqRPMT on server side and the semi-honest security for OT on receiver side, the simulation is indistinguishable to the real view via standard hybrid argument.

This proves the theorem. □

8.1 Private-ID

Recently, Buddhavarapu et al. [BKM+20] propose a two-party functionality called private-ID, which assigns two parties, each holding a set of items, a truly random identifier per item (where identical items receive the same identifier). As a result, each party obtains identifiers to his own set, as well as identifiers associated the union of their input sets. With private-ID, two parties can sort their private set with respect to a global set of identifiers, and then can proceed item-by-item, doing any desired private computation, being assured that identical items are aligned. Buddhavarapu et al. [BKM+20] also give a concrete DDH-based private-ID protocol. Garimella et al. [GMR+21] show how to build private-ID from oblivious PRF and PSU. Roughly speaking, their approach proceeds in two phases. In phase 1, P_1 holding X and P_2 holding Y run an OPRF twice by switching the role, so that first P_1 learns k_1 and P_2 learns $F_{k_1}(y_i)$, and second P_2 learns k_2 and P_1 learns $F_{k_2}(x_i)$. The random identifier of an item z is thus defined as $id_z = F_{k_1}(z) \oplus F_{k_2}(z)$. After phase 1, both parties can compute identifiers for their own items. In phase 2, they simply engage a PSU protocol on their sets $id(X)$ and $id(Y)$ to finish private-ID.

Our method is largely inspired by the approach presented in [GMR⁺21]. We first observe that in phase 1, the two parties essentially need to engage a distributed OPRF protocol, as we formally depict in Figure 13. The random identifier of an item z is defined as $G_{k_1,k_2}(z)$, where G is a PRF determined by key (k_1, k_2). Furthermore, note that $id(X)$ and $id(Y)$ are pseudorandom, which means in phase 2 a distributional PSU protocol suffices, whose semi-honest security is additionally defined on the input distribution. Looking ahead, such relaxation may lead to nice efficiency improvement.

In this work, we instantiate the generic private-ID construction as follows. We first realize the distributed OPRF protocol by running the multi-point OPRF [CM20] twice in reverse order. We then run the PSU protocol from cwPRF-based mqRPMT with the obtained two sets of pseudorandom identifiers as input to fulfill the private-ID functionality.

Parameters:

- Common input: PRF $G : K \times D \rightarrow R$, where $K = K_1 \times K_2$.
- Input of P_1: $X = \{x_1, \ldots, x_{n_1}\} \in D$.
- Input of P_2: $Y = \{y_1, \ldots, y_{n_2}\} \in D$.

Protocol:

1. P_1 obtains $\{G_{k_1,k_2}(x_i))\}_{i \in [n_1]}$ and k_1.
2. P_2 obtains $\{G_{k_1,k_2}(y_i))\}_{i \in [n_2]}$ and k_2,

Figure 13: Ideal functionality for distributed OPRF

Distributional PSU. Standard security notions for MPC are defined w.r.t. any private inputs. This treatment facilitates secure composition of different protocols. We find that in certain settings it is meaningful to consider weaker security notion by allowing the real-ideal indistinguishability also base on the distribution of private inputs. This is because such relaxed security suffices if the protocol's input is another protocol's output which obeys some distribution, and the relaxation may admit efficiency improvement. Suppose choosing the DDH-based distributed OPRF and DDH-based PSU in the same elliptic curve (EC) group as ingredients, faithful implementation according to the above recipe requires $4n$ hash-to-point operations. Observed that the output of distributed DDH-based OPRF are already pseudorandom EC points. In this case, it suffice to use distributional DDH-based PSU instead, and thus can saving $2n$ hash-to-point operations, which are costly in real-world implementation.

9 Performance

We describe details of our implementation and report the performance of the following set operations: (1) **psi**: intersection of the sets; (2) **card**: cardinality of the intersection; (3) **card-sum**: sum of the associated values for every item in the intersection with cardinality; (4) **psu**: union of the sets; (5) **private-ID**: a universal identifier for every item in the union. We compare our work with the current fastest known protocol implementation for each functionality.

9.1 Implementation Details

Our protocols are written in C++ with detailed documentations, which can be found at `https://github.com/yuchen1024/Kunlun/mpc`. In consistency with our paper, we implement our protocols in a modular fashion. We first implement the core mqRPMT protocol, then build various PSO protocols upon it. Our implementation only relies on the OpenSSL library [Ope], and it can smoothly run on both Linux and MacOS. In contrast, most existing PSO programs rely on multiple libraries and require sophisticated parameters tuning, while sometimes the optimized parameters setting are not explicitly given. Thereby, even running these programs successfully on the same environment would require tremendous efforts.

9.2 Experimental Setup

We run all our protocols and related protocols on Ubuntu 20.04 with a single Intel i7-11700 2.50 GHz CPU (8 physical core) and 16 GB RAM. We simulate the network connection using Linux `tc` command. For the WAN setting, we set the average RTT to be 80 ms and bandwidth to be 50 Mbps. We use a `tc` sub-command to compute the communication complexity, and use running time to compute the computation complexity, which is the maximal time from protocol begin to end, including the messages transmission time.

For a fair comparison, we stick to the following setting for all protocols being evaluated:

- We set the computational security parameter $\kappa = 128$ and the statistical security parameter $\lambda = 40$.
- We set the input set size $n_1 = n_2$ (balanced setting), and randomly generate two input sets conditioned on the intersection size is roughly $0.5n$. We set the item length to be 128 bits. The exception is the protocol in [GMR+21], whose item length is fixed as 61 bits.
- We choose `nist P-256` (also known as `secp256r1` and `prime256v1`) as the common elliptic curve for all ECC-based protocols. The exception is the private-ID protocol in [BKM+20], which heavily relies on the highly-optimized Dalek library implementing efficient ECC operations for `Curve25519`. We mark the ECC-based protocols using point compression technique with ▼.

9.3 Evaluation of Our Core Protocol

We first report the performances of our core protocol cwPRF-based mqRPMT described in Section 5.3, which dominates the communication and computation overheads of its enabling PSO protocols. The experimental results in Table 2 is consistent with theoretical analysis, that is, cwPRF-based mqRPMT is scalable and highly parallelable.

Table 2: The computation and communication complexity of mqRPMT

Protocol	T	Running time (s)						Commu. (MB)		
		LAN			WAN			total		
		2^{12}	2^{16}	2^{20}	2^{12}	2^{16}	2^{20}	2^{12}	2^{16}	2^{20}
mqRPMT	1	0.50	7.20	114.16	1.39	9.68	136.27	0.52	8.35	133.6
	2	0.31	3.89	62.09	1.14	6.54	86.60			
	4	0.22	2.37	40.41	1.11	5.08	62.77			
Speedup		1.6-2.3×	1.9-3.0×	1.8-2.8×	1.2-1.3×	1.5-1.9×	1.6-2.2×	–	–	–
mqRPMT▼	1	0.50	8.00	128.00	1.35	10.15	141.52	0.27	4.35	69.6
	2	0.32	5.05	80.69	1.18	7.11	94.19			
	4	0.23	3.54	58.40	1.08	5.54	71.26			
Speedup		1.6-2.2×	1.6-2.3×	1.6-2.2×	1.1-1.3×	1.4-1.8×	1.5-2×	–	–	–

Table 3: Scaling of cwPRF-based mqRPMT protocol with set size and number of threads. We test our protocol up to 4 threads, since both the server and the client run on a single CPU with 8 physical cores.

We highlight the following two caveats we learned during implementation:

- Point compression is a standard trick used in elliptic-curve cryptography (ECC), which can roughly reduce the storage cost of EC point by half, at the cost of performing decompression when needed. Point decompression was empirically thought to be cheap, but experiment indicates that it could be as expensive as point multiplication. Our perspective is that point compression enables a natural trade-offs between communication and computation. As pointed out in [IKN+20], for a secure protocol communication cost is often far more important than computation, especially in case the involving parties cannot be co-located (WAN setting). Looking ahead, experimental results demonstrate that the total running time gives a large weight to communication cost in bandwidth constrained scenarios. Therefore, in the LAN setting we recommend using standard representation, while in the WAN setting we recommend using compressed representation. A quick take-away is that point compression trick pays off in setting where communication is much more expensive than computation.

- The hash to point operation is very tricky in ECC. So far, there is no universal method to securely map arbitrary bit strings to EC points. Here, the vague term secure indicates the hash function could be modeled as a random oracle. A folklore method is the "try-and-increment" algorithm [BLS01], which is also the method adopted in this work. Nevertheless, such hash to point operation could be as expensive as point multiplication, which should be avoid if possible.

9.4 Benchmark Comparison

We derive all kinds of PSO protocols from cwPRF-based mqRPMT protocol, and compare them with the state-of-the-art related protocols. We report the performances for 3 input sizes $n = \{2^{12}, 2^{16}, 2^{20}\}$ all executed over a single thread for LAN and WAN configurations. When testing the PSI-card, PSI-card-sum and PSU protocols in [GMR+21], we set the number of mega-bins as $\{1305, 16130, 210255\}$ and the number of items in each mega-bin as $\{51, 62, 72\}$ for set sizes $n = \{2^{12}, 2^{16}, 2^{20}\}$ respectively. These parameter choices have been tested to be much more optimal than their default ones.

PSI. We compare our mqRPMT-based PSI protocol to the classical DDH-based PSI protocol reported in [PRTY19]. We remark that both of the two PSI protocols in comparison are not competitive with the state-of-the-art PSI protocol. We include them merely for illustrative purposes. PSI protocols from public-key techniques are thought to be very inefficient. Our experiment demonstrates that by choosing modern crypto library they could be practical. Our protocol is roughly an order of magnitude faster than the DDH-based PSI protocol implemented in [PRTY19].

PSI	Running time (s)						Comm. (MB)		
	LAN			WAN			total		
	2^{12}	2^{16}	2^{20}	2^{12}	2^{16}	2^{20}	2^{12}	2^{16}	2^{20}
[PRTY19]	5.07	80.89	1296.67	5.27	81.79	1355.42	0.30	4.74	76.60
Our PSI	**0.50**	**7.24**	**114.66**	**1.71**	**10.50**	**142.45**	0.67	10.38	165.77
Our PSI$^{\blacktriangledown}$	**0.55**	**8.04**	**128.18**	**1.73**	**11.02**	**148.18**	**0.41**	**6.38**	**101.63**

Table 4: Communication cost and running time of PSI protocol.

PSI-card. We compare our mqRPMT-based PSI-card protocol to the PSI-card protocol in [GMR+21]. Our protocol achieves a $1.17 - 6.62\times$ speedup in running time, and reduces the communication cost by a factor $10.85 - 14.80\times$.

PSI-card	Running time (s)						Comm. (MB)		
	LAN			WAN			total		
	2^{12}	2^{16}	2^{20}	2^{12}	2^{16}	2^{20}	2^{12}	2^{16}	2^{20}
[GMR+21]	1.00	8.41	126.01	8.60	27.46	323.52	2.93	55.49	1030
Our PSI-card	**0.49**	**7.20**	**114.31**	**1.30**	**9.68**	**136.06**	**0.52**	**8.36**	**133.71**
Our PSI-card$^{\blacktriangledown}$	**0.53**	**8.00**	**128.00**	**1.35**	**10.16**	**141.31**	**0.27**	**4.35**	**69.6**

Table 5: Communication cost and running time of PSI-card protocol.

PSI-card-sum. We compare our mqRPMT-based PSI-card-sum protocol to the PSI-card-sum protocol (the most efficient and also the deployed one based on DH-protocol+Paillier) in [IKN+20]. We do not compare the protocol described in [GMR+21] since its functionality is not the standard one, as we discussed in the introduction. Our protocol is advantageous than the protocol of [GMR+21] due to our random masking trick is much simpler and efficient than the AHE-based technique. Particularly, the upper bound of intersection sum in [GMR+21] is closely tied to the AHE scheme in use, which requires sophisticated parameter tuning and ciphertext packing techniques. In our protocol, the upper bound of intersection sum can be flexibly adjusted according to applications. As shown in Table 6, our protocol is roughly $8-40\times$ faster than the protocol presented in [IKN$^+$20] (the data marked in gray is obtained on an Intel Xeon E5-1650 3.50 GHz CPU, which is roughly $1.4\times$ faster than our CPU).

PSI-card-sum	Running time (s)						Comm. (MB)		
	LAN			WAN			total		
	2^{12}	2^{16}	2^{20}	2^{12}	2^{16}	2^{20}	2^{12}	2^{16}	2^{20}
[IKN$^+$20] (deployed)	23.64	48.93	776.46	30.10	–	–	2.72	5.1	84
Our PSI-card-sum	**0.51**	**7.22**	**113.66**	**1.46**	**9.68**	**136.27**	**0.64**	**9.89**	**157.80**
Our PSI-card-sum$^\blacktriangledown$	**0.57**	**8.12**	**129.66**	**1.94**	**11.83**	**157.66**	**0.38**	**5.87**	**93.74**

Table 6: Communication cost and running time of PSI-card-sum protocol. We assume each associated value is a non-negative integer in $[0, 2^{32})$ conditioned on the upper bound of intersection sum is 2^{32}. We note that the implementation of [IKN$^+$20] only works in our enviroment at set size 2^{12}. For larger set size, we encounter a run time error reported in [Pri], which has not been fixed yet. Hence, we use the data reported in the orginal paper [IKN$^+$20] for set sizes 2^{16} and 2^{20} in the LAN setting, and mark them in gray color. Cells with "–" denote the corresponding data is not available in [IKN$^+$20].

PSU. We compare our mqRPMT-based PSU protocol to the state-of-the-art PSU protocols in [GMR$^+$21, ZCL$^+$22, JSZ$^+$22]. The work [ZCL$^+$22] provides two PSU protocols from public-key and symmetric-key respectively. The work [JSZ$^+$22] also provides two PSU protocols called PSU-R abd PSU-S. We choose the most efficient PKE-PSU [ZCL$^+$22] and PSU-S [JSZ$^+$22] for comparison. Among all the mentioned PSU protocols, only the PSU protocols in [ZCL$^+$22] and our PSU protocol achieve strict linear communication and computation complexity. The experimental results in Table 7 indicate that our PSU protocol is also the most communication efficient one. Comparing to the most concrete efficient PSU protocol of [JSZ$^+$22], our protocol is $8.75-13.2\times$ smaller in terms of communication cost, and thus achieves a $1.93-10.98\times$ speedup in the moderate WAN setting.

PSU	Running time (s)						Comm. (MB)		
	LAN			WAN			total		
	2^{12}	2^{16}	2^{20}	2^{12}	2^{16}	2^{20}	2^{12}	2^{16}	2^{20}
[GMR$^+$21]	1.16	10.06	151.34	10.34	38.52	349.43	3.85	67.38	1155
[ZCL$^+$22]	4.87	12.19	141.38	5.78	15.75	182.88	1.35	21.41	342.38
[ZCL$^+$22]$^\blacktriangledown$	5.10	15.13	187.29	5.82	17.37	210.06	0.77	12.20	195.17
[JSZ$^+$22]	2.04	3.94	36.67	18.68	27.00	276.99	3.59	70.37	1341.55
Our PSU	**0.52**	**7.27**	**114.44**	**1.70**	**10.56**	**143.29**	**0.68**	**10.38**	**165.77**
Our PSU$^\blacktriangledown$	**0.57**	**8.04**	**128.20**	**1.76**	**10.92**	**148.15**	**0.41**	**6.38**	**101.63**

Table 7: Communication cost and running time of PSI-card-sum protocol. We randomly generate two input sets conditioned on the union size is roughly $1.5n$.

Private-ID. We compare our private-ID protocol described in Section 8.1 to the state-of-the-art PSU protocols in [BKM$^+$20, GMR$^+$21]. The experimental results in Table 9 show that our private-ID protocol achieves a $1.39-4.75\times$ speedup comparing to the existing most computation efficient private-ID protocol of [GMR$^+$21], while its bandwidth is only marginally larger than the most communication efficient private-ID protocol [BKM$^+$20]. We believe our protocol is the most efficient one in terms of monetary cost.

Table 8: The computation and communication complexity of Private-ID

Private-ID	Running time (s)						Comm. (MB)		
	LAN			WAN			total		
	2^{12}	2^{16}	2^{20}	2^{12}	2^{16}	2^{20}	2^{12}	2^{16}	2^{20}
[GMR+21]	1.65	11.023	158.76	13.82	43.00	385.12	4.43	76.57	1293
[BKM+20]	2.21	37.56	671.75	7.98	46.97	710.94	1.00	15.97	226.70
Our Private-ID	**0.77**	**8.40**	**114.45**	**2.91**	**13.62**	**148.48**	**1.46**	**20.52**	**330.40**
Our Private-ID$^{\blacktriangledown}$	**0.89**	**9.57**	**146.73**	**3.07**	**14.53**	**186.98**	**1.13**	**16.43**	**266.31**

Table 9: Communication cost and running time of private-ID protocol. We randomly generate two input sets conditioned on the union size is roughly $1.5n$.

10 Summary

In this work, we identify that mqRPMT protocol is a "Swiss Army Knife" for private set operations. By coupling with OT, we create a unified PSO framework from mqRPMT, which can greatly reduce the deployment and maintaining costs of PSO in the real world. We build mqRPMT from two newly introduced cryptographic primitives, namely cwPRF and pOPRF respectively. By instantiating cwPRF and pOPRF from DDH-like assumptions, we obtain mqRPMT protocols with linear complexity. The significance of this result is two folds. The first is of practical interest, namely providing a scalable and easy to implement PSO framework. This framework yields a family of PSO protocols that are competitive or superior to existing ones. Particularly, all our protocols are pretty simple and clean. We view the simplicity as great advantage from practical point of view. The second is of more theoretical interest, namely introducing cwPRF and pOPRF. The notion of cwPRF can be viewed as the right cryptographic abstraction of the celebrated DH functions, which not only demonstrates that the DDH assumption is complete for PSO, but also opens the door for possible new instantiations beyond DDH-like assumptions. The notion of pOPRF is of independent interest. It enriches the OPRF family, and help us to understand which OPRF-based PSI protocols can (or cannot) be adapted to PCSI/PSU protocols. We left more applications and efficient constructions of pOPRF as an interesting problem.

Finally, we present a semi-generic conversion from a category mqPMT protocols called Sigma-mqPMT to mqRPMT, making the first step towards investigating relations between the two core protocols. As an application of such conversion, we obtain a mqRPMT protocol from FHE which is suitable in the unbalanced setting. However, the resulting mqRPMT is a slightly weak version in the sense that the intersection size is leaked to the sender. We left the construction of standard mqRPMT in the unbalanced setting as an open problem.

Acknowledgments

We thank Yilei Chen for bringing up EGA to our attention, and thank Navid Alamati for helpful clarification on input-homomorphic weak PRF. We thank Hong Cheng for enlightening discussion on OPRF. We particularly thank Weiran Liu for many instructive advice on both theoretical constructions and implementation techniques.

References

[AES03] Rakesh Agrawal, Alexandre V. Evfimievski, and Ramakrishnan Srikant. Information sharing across private databases. In *2003 ACM SIGMOD International Conference on Management of Data*, pages 86–97. ACM, 2003.

[AFMP20] Navid Alamati, Luca De Feo, Hart Montgomery, and Sikhar Patranabis. Cryptographic group actions and applications. In *Advances in Cryptology - ASIACRYPT 2020*, volume 12492 of *LNCS*, pages 411–439. Springer, 2020.

[ALSZ15] Gilad Asharov, Yehuda Lindell, Thomas Schneider, and Michael Zohner. More efficient oblivious transfer extensions with security for malicious adversaries. In *Advances in Cryptology - EUROCRYPT 2015*, volume 9056 of *Lecture Notes in Computer Science*, pages 673–701. Springer, 2015.

[BKM+20] Prasad Buddhavarapu, Andrew Knox, Payman Mohassel, Shubho Sengupta, Erik Taubeneck, and Vlad Vlaskin. Private matching for compute. 2020. https://eprint.iacr.org/2020/599.

[BLS01] Dan Boneh, Ben Lynn, and Hovav Shacham. Short signatures from the weil pairing. In *Advances in Cryptology - ASIACRYPT 2001*, volume 2248 of *LNCS*, pages 514–532, 2001.

[CHLR18] Hao Chen, Zhicong Huang, Kim Laine, and Peter Rindal. Labeled PSI from fully homomorphic encryption with malicious security. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS 2018*, pages 1223–1237. ACM, 2018.

[CLR17] Hao Chen, Kim Laine, and Peter Rindal. Fast private set intersection from homomorphic encryption. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS 2017*, pages 1243–1255. ACM, 2017.

[CM20] Melissa Chase and Peihan Miao. Private set intersection in the internet setting from lightweight oblivious PRF. In *Advances in Cryptology - CRYPTO 2020*, volume 12172 of *Lecture Notes in Computer Science*, pages 34–63. Springer, 2020.

[CMdG+21] Kelong Cong, Radames Cruz Moreno, Mariana Botelho da Gama, Wei Dai, Ilia Iliashenko, Kim Laine, and Michael Rosenberg. Labeled PSI from homomorphic encryption with reduced computation and communication. In *CCS '21: 2021 ACM SIGSAC Conference on Computer and Communications Security*, pages 1135–1150. ACM, 2021.

[DC17] Alex Davidson and Carlos Cid. An efficient toolkit for computing private set operations. In *Information Security and Privacy - 22nd Australasian Conference, ACISP 2017*, volume 10343 of *Lecture Notes in Computer Science*, pages 261–278. Springer, 2017.

[DCW13] Changyu Dong, Liqun Chen, and Zikai Wen. When private set intersection meets big data: an efficient and scalable protocol. In *CCS 2013*, pages 789–800, 2013.

[FIPR05] Michael J. Freedman, Yuval Ishai, Benny Pinkas, and Omer Reingold. Keyword search and oblivious pseudorandom functions. In *Theory of Cryptography, Second Theory of Cryptography Conference, TCC 2005*, volume 3378 of *Lecture Notes in Computer Science*, pages 303–324. Springer, 2005.

[Fri07] Keith B. Frikken. Privacy-preserving set union. In *Applied Cryptography and Network Security, 5th International Conference, ACNS 2007*, volume 4521 of *Lecture Notes in Computer Science*, pages 237–252. Springer, 2007.

[GGM86] Oded Goldreich, Shafi Goldwasser, and Silvio Micali. How to construct random functions. *J. ACM*, 33(4):792–807, 1986.

[GMR+21] Gayathri Garimella, Payman Mohassel, Mike Rosulek, Saeed Sadeghian, and Jaspal Singh. Private set operations from oblivious switching. In *Public-Key Cryptography - PKC 2021*, volume 12711 of *Lecture Notes in Computer Science*, pages 591–617. Springer, 2021.

[GPR+21] Gayathri Garimella, Benny Pinkas, Mike Rosulek, Ni Trieu, and Avishay Yanai. Oblivious key-value stores and amplification for private set intersection. In *Advances in Cryptology - CRYPTO 2021*, volume 12826 of *Lecture Notes in Computer Science*, pages 395–425. Springer, 2021.

[HFH99] Bernardo A. Huberman, Matthew K. Franklin, and Tad Hogg. Enhancing privacy and trust in electronic communities. In *Proceedings of the First ACM Conference on Electronic Commerce (EC-99)*, pages 78–86. ACM, 1999.

[HN10] Carmit Hazay and Kobbi Nissim. Efficient set operations in the presence of malicious adversaries. In *Public Key Cryptography - PKC 2010*, volume 6056 of *Lecture Notes in Computer Science*, pages 312–331. Springer, 2010.

[IKN+20] 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 *IEEE European Symposium on Security and Privacy, EuroS&P 2020*, pages 370–389. IEEE, 2020.

[IKNP03] Yuval Ishai, Joe Kilian, Kobbi Nissim, and Erez Petrank. Extending oblivious transfers efficiently. In *Advances in Cryptology - CRYPTO 2003*, volume 2729 of *Lecture Notes in Computer Science*, pages 145–161. Springer, 2003.

[JSZ+22] Yanxue Jia, Shi-Feng Sun, Hong-Sheng Zhou, Jiajun Du, and Dawu Gu. Shuffle-based private set union: Faster and more secure. In *USENIX 2022*, 2022.

[KK13] Vladimir Kolesnikov and Ranjit Kumaresan. Improved OT extension for transferring short secrets. In *Advances in Cryptology - CRYPTO 2013*, volume 8043 of *Lecture Notes in Computer Science*, pages 54–70. Springer, 2013.

[KKRT16] Vladimir Kolesnikov, Ranjit Kumaresan, Mike Rosulek, and Ni Trieu. Efficient batched oblivious PRF with applications to private set intersection. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, CCS 2016*, pages 818–829. ACM, 2016.

[KRTW19] Vladimir Kolesnikov, Mike Rosulek, Ni Trieu, and Xiao Wang. Scalable private set union from symmetric-key techniques. In *Advances in Cryptology - ASIACRYPT 2019*, volume 11922 of *Lecture Notes in Computer Science*, pages 636–666. Springer, 2019.

[KS05] Lea Kissner and Dawn Xiaodong Song. Privacy-preserving set operations. In *Advances in Cryptology - CRYPTO 2005*, volume 3621 of *Lecture Notes in Computer Science*, pages 241–257. Springer, 2005.

[Mea86] Catherine A. Meadows. A more efficient cryptographic matchmaking protocol for use in the absence of a continuously available third party. In *Proceedings of the 1986 IEEE Symposium on Security and Privacy*, pages 134–137. IEEE Computer Society, 1986.

[MPR$^+$20] Peihan Miao, Sarvar Patel, Mariana Raykova, Karn Seth, and Moti Yung. Two-sided malicious security for private intersection-sum with cardinality. In *Advances in Cryptology - CRYPTO 2020*, volume 12172 of *Lecture Notes in Computer Science*, pages 3–33. Springer, 2020.

[NPR99] Moni Naor, Benny Pinkas, and Omer Reingold. Distributed pseudo-random functions and kdcs. In *Advances in Cryptology - EUROCRYPT 1999*, volume 1592 of *Lecture Notes in Computer Science*, pages 327–346. Springer, 1999.

[NR95] Moni Naor and Omer Reingold. Synthesizers and their application to the parallel construction of psuedo-random functions. In *36th Annual Symposium on Foundations of Computer Science, FOCS 1995*, pages 170–181. IEEE Computer Society, 1995.

[Ope] https://github.com/openssl.

[Pri] https://github.com/google/private-join-and-compute/issues/16.

[PRTY19] Benny Pinkas, Mike Rosulek, Ni Trieu, and Avishay Yanai. Spot-light: Lightweight private set intersection from sparse OT extension. In *Advances in Cryptology - CRYPTO 2019*, volume 11694 of *Lecture Notes in Computer Science*, pages 401–431. Springer, 2019.

[PSZ14] Benny Pinkas, Thomas Schneider, and Michael Zohner. Faster private set intersection based on OT extension. In *Proceedings of the 23rd USENIX Security Symposium, 2014*, pages 797–812. USENIX Association, 2014.

[PSZ18] Benny Pinkas, Thomas Schneider, and Michael Zohner. Scalable private set intersection based on OT extension. *ACM Trans. Priv. Secur.*, 21(2):7:1–7:35, 2018.

[Rab] Michael O. Rabin. How to exchange secrets with oblivious transfer. http://eprint.iacr.org/2005/187.

[RR17] Peter Rindal and Mike Rosulek. Improved private set intersection against malicious adversaries. In *Advances in Cryptology - EUROCRYPT 2017*, volume 10210 of *LNCS*, pages 235–259, 2017.

[RS21] Peter Rindal and Phillipp Schoppmann. VOLE-PSI: fast OPRF and circuit-psi from vector-ole. In *Advances in Cryptology - EUROCRYPT 2021*, volume 12697 of *Lecture Notes in Computer Science*, pages 901–930. Springer, 2021.

[Sha80] Adi Shamir. On the power of commutativity in cryptography. In *ICALP 1980*, volume 85 of *Lecture Notes in Computer Science*, pages 582–595. Springer, 1980.

[TCLZ22] Binbin Tu, Yu Chen, Qi Liu, and Cong Zhang. Fast unbalanced private set union from fully homomorphic encryption, 2022. https://eprint.iacr.org/2022/653.

[ZCL$^+$22] Cong Zhang, Yu Chen, Weiran Liu, Min Zhang, and Dongdai Lin. Optimal private set union from multi-query reverse private membership test, 2022. https://eprint.iacr.org/2022/358.

A Missing Definitions

A.1 Weak Pseudorandom EGA

We begin by recalling the definition of a group action.

Definition A.1 (Group Actions)**.** A group $\mathbb{G}$ is said to *act on* a set X if there is a map $\star : \mathbb{G} \times X \to X$ that satisfies the following two properties:

1. Identity: if e is the identity element of $\mathbb{G}$, then for any $x \in X$, we have $e \star x = x$.
2. Compatibility: for any $g, h \in \mathbb{G}$ and any $x \in X$, we have $(gh) \star x = g \star (h \star x)$.

From now on, we use the abbreviated notation $(\mathbb{G}, X, \star)$ to denote a group action. If $(\mathbb{G}, X, \star)$ is a group action, for any $g \in \mathbb{G}$ the map $\phi_g : x \mapsto g \star x$ defines a permutation of X.

We then define an effective group action (EGA) [AFMP20] as follows.

Definition A.2 (Effective Group Actions)**.** A group action $(\mathbb{G}, X, \star)$ is *effective* if the following properties are satisfied:

1. The group $\mathbb{G}$ is finite and there exist PPT algorithms for:
   (a) Membership testing, i.e., to decide if a given bit string represents a valid group element in $\mathbb{G}$.
   (b) Equality testing, i.e., to decide if two bit strings represent the same group element in $\mathbb{G}$.
   (c) Sampling, i.e., to sample an element g from a uniform (or statistically close to) distribution on $\mathbb{G}$.
   (d) Operation, i.e., to compute gh for any $g, h \in \mathbb{G}$.
   (e) Inversion, i.e., to compute g^{-1} for any $g \in \mathbb{G}$.
2. The set X is finite and there exist PPT algorithms for:
   (a) Membership testing, i.e., to decide if a bit string represents a valid set element.
   (b) Unique representation, i.e., given any set element $x \in X$, compute a string $\hat{x}$ that canonically represents x.
3. There exists a distinguished element $x_0 \in X$, called the origin, such that its bit-string representation is known.
4. There exists an efficient algorithm that given (some bit-string representations of) any $g \in \mathbb{G}$ and any $x \in X$, outputs $g \star x$.

Definition A.3 (Weak Pseudorandom EGA)**.** A group action $(G, X, \star)$ is weakly pseudorandom if the family of efficiently commutable permutation $\{\phi_g : X \to X\}_{g \in G}$ is weakly pseudorandom, i.e., there is no PPT adversary that can distinguish tuples of the form $(x_i, g \star x_i)$ from (x_i, u_i) where $g \xleftarrow{\text{R}} \mathbb{G}$ and each $x_i, u_i \xleftarrow{\text{R}} X$.

B Instantiations of Sigma-mqPMT

B.1 Sigma-mqPMT from DDH

We first present an instantiation of Sigma-mqPMT based on the DDH assumption, which is obtained by plugging DDH-based OPRF to the above generic construction.

$$
\begin{array}{ccc}
P_1 \text{ (server)} & & P_2 \text{ (client)} \\
Y = (y_1, \ldots, y_n) & & X = (x_1, \ldots, x_n) \\[1em]
k \xleftarrow{\text{R}} \mathbb{Z}_p & \xrightarrow{\quad a \leftarrow \{\mathsf{H}(y_1)^k, \ldots, \mathsf{H}(y_{n_1})^k\} \quad} & \\[1em]
 & \xleftarrow{\quad \vec{q} = \{q_1, \ldots, q_{n_2}\} \quad} & r \xleftarrow{\text{R}} \mathbb{Z}_p,\ q_i \leftarrow \mathsf{H}(x_i)^r \\[1em]
z_i \leftarrow (\mathsf{H}(x_i)^r)^k & \xrightarrow{\quad \vec{z} = \{z_1, \ldots, z_{n_2}\} \quad} & e_i := z_i \in a
\end{array}
$$

B.2 Sigma-mqPMT from FHE

We then present an instantiation of Sigma-mqPMT based on oblivious polynomial evaluation (OPE). By instantiating OPE from FHE, we obtain the following mqPMT protocol, which is the backbone of [CLR17].

$$\begin{tikzcd}[column sep=huge]
\begin{array}{c} P_1 \text{ (server)} \\ Y=(y_1,\dots,y_{n_1}) \end{array} & & \begin{array}{c} P_2 \text{ (client)} \\ X=(x_1,\dots,x_{n_2}) \end{array} \\
a \leftarrow \perp \arrow[rr, dashed, "\perp"] & & {} \\
{} & & q_i \leftarrow \mathsf{FHE.Enc}(pk, x_i) \arrow[ll, "\vec{q}=\{q_1,\dots,q_{n_2}\}"'] \\
\begin{array}{r} r_i \xleftarrow{\text{R}} \mathbb{F} \\ f_i \leftarrow r_i \prod_{y \in Y}(y_i - x) \\ z_i \leftarrow \mathsf{FHE.Eval}(pk, f_i, q_i) \end{array} \arrow[rr, "\vec{z}=\{z_1,\dots,z_{n_2}\}"] & & e_i := \mathsf{FHE.Dec}(dk, z_i) \overset{?}{=} 0
\end{tikzcd}$$

Alternatively, we can realize OPE from additively homomorphic encryption. The change is that each q_i now consists of n_1 ciphertexts of the following form: $\{\mathsf{AHE.Enc}(pk, x_i^1), \dots, \mathsf{AHE.Enc}(pk, x_i^{n_1})\}$.

Remark B.1. As noted in [CLR17], the above protocol only serves as a toy example to illustrate the idea of how to using FHE to build PSI, which is impractical. They also show how to make the basic protocol efficient. However, the optimizing techniques destroy structure and properties of Sigma-mqPMT. As a consequence, so far the transformation from Sigma-mqPMT to mqRPMT* does not have efficient instantiation in the unbalanced setting, and only serves as a proof of concept.

C Missing Security Proofs

C.1 Proof of Permuted OPRF Based on the DDH Assumption

Theorem C.1. *The permuted OPRF protocol described in Figure 8 is secure in the semi-honest model assuming* H *is a random oracle and the DDH assumption holds.*

Proof. We exhibit simulators Sim_{P_1} and Sim_{P_2} for simulating corrupt P_1 and P_2 respectively, and argue the indistinguishability of produced transcript from the real execution.

Security against corrupt receiver. Sim_{P_2} simulates the view of corrupt receiver P_2, which consists of P_2's randomness, input, output and received messages. We formally show Sim_{P_2}'s simulation is indistinguishable from the real execution via a sequence of hybrid transcripts.

Hybrid$_0$: P_2's view in the real protocol.

Hybrid$_1$: Given P_2's input $X = (x_1, \dots, x_n)$ and output $\{y_{\pi(1)}, \dots, y_{\pi(n)}\}$, Sim_{P_2} emulates the random oracle H honestly, picks $s \xleftarrow{\text{R}} \mathbb{Z}_p$, simulates message from P_1 as $\{y^s_{\pi(1)}, \dots, y^s_{\pi(n)}\}$.

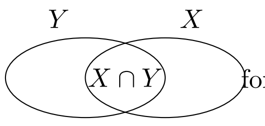
Clearly, Sim_{P_2}'s simulated view is identical to the real view.

Security against corrupt sender. Sim_{P_1} simulates the view of corrupt sender P_1, which consists of P_1's randomness, input, output and received messages. We formally show Sim_{P_1}'s simulation is indistinguishable from the real execution via a sequence of hybrid transcripts,

Hybrid$_0$: P_1's view in the real protocol.

Hybrid$_1$: Given P_1's output k and π, Sim_{P_1} chooses the randomness s for P_2, and simulates with the knowledge of $X = (x_1, \dots, x_n)$:

- RO queries: Sim_{P_1} honestly emulates random oracle H. For every query $\langle z_i \rangle$, picks $\alpha_i \xleftarrow{\text{R}} \mathbb{G}$ and assigns $\mathsf{H}(z_i) := \alpha_i$.
- Sim_{P_1} outputs $(\beta_1^s, \dots, \beta_n^s)$, where $\mathsf{H}(x_i) = \beta_i$.



for $z_i \in \{0,1\}^\ell$, $\mathsf{H}(z_i) := \alpha_i \xleftarrow{\text{R}} \mathbb{G}$

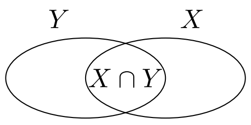
Clearly, Sim_{P_1}'s simulated view in Hybrid$_1$ is identical to P_1's real view.

Hybrid$_2$: Sim_{P_1} does not choose the randomness for P_2, and simulates without the knowledge of X. It honestly emulates random oracle H as in Hybrid$_1$, and only changes the simulation of P_2's message.

- Sim_{P_1} outputs $(g^{c_1}, \dots, g^{c_n})$ where $c_i \xleftarrow{\text{R}} \mathbb{Z}_p$.

We argue that the view in Hybrid$_1$ and Hybrid$_2$ are computationally indistinguishable. Let $\mathcal{A}$ be a PPT adversary against the DDH assumption. Given the DDH challenge $g^a, g^{b_1}, \dots, g^{b_n}, g^{c_1}, \dots, g^{c_n})$ where $a, b_i \xleftarrow{\text{R}} \mathbb{Z}_p$, $\mathcal{A}$ is asked to distinguish if $c_i = ab_i$ or random values. $\mathcal{A}$ implicitly sets P_2's randomness $s := a$, and simulates (with the knowledge of X) as below:

- RO queries: for each query $\langle z_i \rangle$, if $z_i \notin X$, picks $\alpha_i \xleftarrow{\text{R}} \mathbb{G}$ and assigns $\mathsf{H}(z_i) := \alpha_i$; if $z_i \in X$, assigns $\mathsf{H}(x_i) := g^{b_i}$.
- Outputs $(g^{c_1}, \dots, g^{c_n})$.



for $z_i \notin X$, $\mathsf{H}(z_i) := \alpha_i \xleftarrow{\text{R}} \mathbb{G}$
for $z_i \in X$, $\mathsf{H}(z_i) := g^{b_i}$

Clearly, if $c_i = ab_i$, $\mathcal{A}$ simulates Hybrid$_1$. Else, it simulates Hybrid$_2$. Thereby, Sim_{P_1}'s simulated view is computationally indistinguishable to P_1's real view.

This proves the theorem. □

Remark C.1. In the above security proof, when establishing the security against corrupt sender, we can obtain a more modular proof by reducing the indistinguishability of simulated views in Hybrid$_1$ and Hybrid$_2$ to the pseudorandomness of $F_k(\mathsf{H}(\cdot))$, which is in turn based on the DDH assumption.