Private Set Operations from Multi-Query Reverse Private Membership Test

Yu Chen * Min Zhang* Cong Zhang † Minglang Dong* Weiran Liu ‡

Abstract

Private set operations allow two parties to perform secure computation on their private sets, including intersection, union and functions of intersection/union. In this paper, we put forth a framework to perform private set operations. The technical core of our framework is the multi-query reverse private membership test (mqRPMT) protocol [ZCL+23], in which a client with a vector $X = (x_1, \ldots, x_n)$ interacts with a server holding a set Y, and eventually the server learns only a bit vector $(e_1, \ldots, e_n)$ indicating whether $x_i \in Y$ without learning the value of x_i, while the client learns nothing. We present two constructions of mqRPMT from newly introduced cryptographic notions, one is based on commutative weak pseudorandom function (cwPRF), and the other is based on permuted oblivious pseudorandom function (pOPRF). Both cwPRF and pOPRF can be realized from the decisional Diffie-Hellman (DDH)-like assumptions in the random oracle model. We also introduce a slightly weaker version of mqRPMT dubbed mqRPMT*, in which the client also learns the cardinality of $X \cap Y$. We show that mqRPMT* can be built 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 into the framework, we obtain various PSO protocols that are superior or competitive to the state-of-the-art protocols. For intersection functionality, our protocol is faster than the most efficient one for small sets. For cardinality functionality, our protocol achieves a $2.4 - 10.5\times$ speedup and a $10.9 - 14.8\times$ shrink in communication cost. For cardinality-with-sum functionality, our protocol achieves a $28.5 - 76.3\times$ speedup and $7.4\times$ shrink in communication cost. For union functionality, our protocol is the first one that attains strict linear complexity, and requires the lowest concrete computation and communication costs in all settings, achieving a $2.7-17\times$ speedup and about $2\times$ shrink in communication cost. Specifically, for input sets of size 2^{20}, our PSU protocol requires roughly 100 MB of communication and 16 seconds using 4 threads on a laptop in the LAN setting. Our improvement on PSU also translates to related functionality, yielding the most efficient private-ID protocol to date. 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
‡Alibaba Group. Email: weiran.lwr@alibaba-inc.com

Contents

1 Introduction

Consider several parties, each with a private set of items, want to perform computation on their private sets without revealing any other information to each other. Private set operation (PSO) refers to such family of interactive cryptographic protocols that fulfill this task, which take private sets as inputs and compute the desired function, delivering the result to the participants. In this work, we focus on two-party PSO protocols with semi-honest security. In what follows, we briefly review related works in terms of typical functionalities.

Private set intersection (PSI). PSI allows two parties, the sender and the receiver, to compute the intersection of their private sets X and Y, such that the receiver only learns $X \cap Y$ and the sender learns nothing. PSI has found numerous applications including privacy-preserving location sharing [NTL+11], private contact discovery [DRRT18], DNA testing and pattern matching [TKC07]. 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 implementations. The most efficient PSI protocols [KKRT16, PRTY19, CM20, GPR+21, RS21] mainly rely on symmetric-key operations, except $O(\kappa)$ public-key operations (where κ is a computational security parameter) in base OT used in the OT extension protocol. We refer to [PSZ18] for a good survey of different PSI paradigms.

Private computing on set intersection (PCSI). Certain real-world application scenarios only require partial/aggregated information about the intersection. In this setting fine-grained private computation on set intersection (PCSI) is needed, such as PSI-card for intersection cardinality [HFH99, AES03, CGT12], PSI-card-sum for intersection cardinality and sum [IKN+20, GMR+21]. For general-purpose PCSI (also known as circuit-PSI) [HEK12, PSTY19], the parties learn secret shares of elements in the set intersection, which can be further fed into generic 2PC to compute $g(X \cap Y)$ for arbitrary function g.

Private set union (PSU). PSU allows two parties, the sender and the receiver, to compute the union of their private sets X and Y, such that the receiver only learns $X \cup Y$ and the sender learns nothing. Like PSI, PSU also has many applications in practice, such as cyber risk assessment and management [LV04], IP blacklist and vulnerability data aggregation [HLS+16], private DB supporting full join [KRTW19] and private ID [GMR+21]. Existing PSU protocols can be broadly divided into two categories based on the underlying cryptographic techniques used. The first category mainly relies on public-key techniques [KS05, Fri07, HN10, DC17], while the second category mainly relies on symmetric-key techniques [KRTW19, GMR+21, JSZ+22]. We refer to [ZCL+23] for a comprehensive survey of existing PSU protocols.

PSO protocols are primarily designed for the balanced setting, where the sizes of two sets are approximately the same. Recently, some works have started considering the unbalanced setting, where one set is much larger than the other. Among PSO protocols, PSI has been extensively studied. In the balanced setting, numerous PSI protocols achieve linear complexity, and the current state-of-the-art PSI [RR22] is almost as efficient as the naive insecure hash-based protocol. In the unbalanced setting, a series of work [CLR17, CHLR18, CMdG+21] show how to leverage fully homomorphic encryption (FHE) to build PSI protocols with sublinear complexity in the larger set size. In contrast, the study of PCSI and PSU is less satisfactory. In the case of PCSI, while a few protocols [PSTY19, IKN+20] achieve linear complexity in the balanced setting, their practical performance is poor. As shown in [GMR+21], even in the simplest case of semi-honest PCSI – like PSI-card – is concretely about 20× slower and requires over 30× more communication than PSI. [CHLR18] proposed PSI-card and PSI-card-sum protocols based on generic 2PC in the unbalanced setting, but these protocols are more of theoretical interest, and are not accompanied by implementations. In the case of PSU, no protocol with linear complexity in either balanced or unbalanced setting is known for a long time being. It is until very recently, Zhang et al. [ZCL+23] make a breakthrough by proposing the first PSU with linear complexity. However, their work does not close this issue. Their concrete PSU protocols have a large constant term in computation complexity, incurring a significant efficiency gap compared with PSI: roughly 25× slower and requires at least 3× more communication than PSI.

It is somewhat surprising that different PSO protocols have significantly different efficiency. One may wonder: what is the reason for this discrepancy? Observe that PSI can be essentially viewed as multi-query private membership test (mqPMT), which has efficient realizations in both balanced and unbalanced settings. However, mqPMT generally does not imply PCSI or PSU. The reason is that mqPMT reveals information about the intersection, which should be hidden from the receiver in PCSI and PSU.

1.1 Motivation

Our motivation of this work is threefold. First, the above discussion indicates that the most efficient PSI protocols may not be easily adapted to PCSI and PSU protocols. Consequently, constructions of different PSO protocols differ vastly in the types of techniques they employ, requiring significant engineering effort and making it difficult to deploy PSO systematically. This calls for a modular approach that allows for an easier navigation in the huge design space. We are thus motivated to seek for a common principal that enables all private set operations through a unified framework. Second, given the large efficiency gap between PSI and other related protocols, we are also motivated to give efficient instantiations of the framework to narrow the gap. Last but not least, it is worth noting that the seminal PSI protocol, DH-PSI [Mea86] (related ideas were appeared in [Sha80, HFH99]), which was derived from the Diffie-Hellman key-exchange protocol, based on the decisional Diffie-Hellman (DDH) assumption, is still the most easily understood and implemented one among many PSI protocols for over four decades. Somewhat surprisingly, no PSU counterpart of DH-PSI has been discovered yet. It is curious to know whether the DDH assumption is broadly useful in the PSU setting. In summary, our work seeks to address the following questions:

Is there a central building block that enables a unified framework for all private set operations? If so, can we give efficient instantiations with optimal asymptotic complexity and good concrete efficiency? And finally, can the DDH assumption be used to construct efficient PSU protocols?

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. mqRPMT already implies PSI-card by itself; by coupling with OT, mqRPMT implies PSI and PSU; by additionally coupling with simple secret sharing, mqRPMT implies PSI-card-sum and PSI-card-secret-sharing, where the latter further admits general-purpose PCSI with cardinality. Therefore, mqRPMT enables a unified PSO framework, which can perform a variety of private set operations in a flexible manner.

Efficient construction of mqRPMT. We present two generic constructions of mqRPMT. The first is based on a newly formalized cryptographic primitive called commutative weak PRF (cwPRF), while the second is based on a newly introduced 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 constructions with linear communication and computation complexity. Note that the complexity of our PSO framework is dominated by the underlying mqRPMT. Therefore, all resulting PSO protocols inherit optimal linear complexity. Notably, the obtained PSU protocol is arguably the most simple and efficient one among existing PSU 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 relationship between mqRPMT and mqPMT, where the latter is equivalent to PSI. 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 establishing the connection between mqRPMT and mqPMT. Though mqRPMT* deviates from the standard mqRPMT in that it reveals intersection size to the client, this could be a desirable feature in the application scenarios where both parties want to learn intersection size, for example, PSI-card-sum [IKN+20]. We leave the investigation of the general connection between mqPMT and mqRPMT as a challenging open problem.

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

1.3 Technical Overview

PSO from mqRPMT. As discussed above, mqPMT (equivalent to PSI) is generally 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 recently emerged mqRPMT protocol [ZCL+23], which is a generalization of RPMT formalized in [KRTW19], is actually a central protocol underlying all the existing PSU protocols. Roughly speaking, mqRPMT is a two-party protocol between a server holding a set Y and a client holding a vector $X = (x_1, \dots, x_n)$. After the execution, 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 it is the server but not the client learns the test results. This subtle difference turns out to be crucial. To see this, note that in mqRPMT the intersection information (i.e. x_i and e_i) is shared between two parties, while in mqPMT the intersection information is entirely known by the client. In light of this difference, mqRPMT is not only particularly suitable for functionalities that have to keep intersection private, but also retains the necessary information to compute the intersection.

Equipped with the understanding from above, we can build a family of PSO protocols from mqRPMT in a modular fashion. PSI-card protocol is immediate since the cardinality of intersection is exactly the Hamming weight of indication vector. PSI (resp. PSU) protocol can be created by having the receiver (playing the role of server) and the sender (playing the role of client) invoke a mqRPMT protocol in the first place, then carry out n one-sided OTs with $1 - e_i$ (resp. e_i) and x_i. PSI-card-sum and PSI-card-secret-sharing protocols can be constructed by additionally coupling with OT and simple secret-sharing trick. We defer the construction details to Section 8.

Next, we give two generic constructions of mqRPMT. For clarity, we explicitly parameterize RPMT and PMT with two parameters n_1 and n_2, namely (n_1, n_2)-(R)PMT, where n_1 is the size of server's set Y, n_2 is the length of client's vector X, a.k.a. the number of membership test queries.

mqRPMT from cwPRF. Observe that private equality test (PEQT) protocol [PSZ14] not only can be viewed as an extreme case of mqPMT, but also can be viewed as an extreme case of mqRPMT. Under the parameterized notions, PEQT is essentially $(1, 1)$-PMT and $(1, 1)$-RPMT. We choose PEQT as the starting point of our first mqRPMT construction.

The basic idea of building $(1, 1)$-RPMT protocol amenable to extension is using *oblivious joint encoding*, by which an element can only be encoded to a codeword by two parties in a joint manner, while the process reveals nothing to the party without the element. To implement this idea, we formalize a new cryptographic primitive called commutative weak PRF (cwPRF). Let $F : K \times D \to R$ be a family of weak PRF, where $R \subseteq D$. F is commutative if $F_{k_1}(F_{k_2}(x)) = F_{k_2}(F_{k_1}(x))$ for any $k_1, k_2 \in K$ and any $x \in D$. 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, denoted by $\hat{F}$.

Now we are ready to describe the construction of $(1, 1)$-RPMT from cwPRF. The server P_1 holding y and the client P_2 holding x can conduct PEQT functionality via the following steps: (1) P_1 and P_2 generate cwPRF key k_1 and k_2 respectively, and map their items to domain D of F using a common cryptographic hash function H, which will be modeled as a random oracle; (2) P_1 computes and sends $F_{k_1}(\mathsf{H}(y))$ to P_2; (3) P_2 computes and sends $F_{k_2}(\mathsf{H}(x))$ and $F_{k_2}(F_{k_1}(\mathsf{H}(y)))$ to P_1; (4) 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 more than the test result. In the above construction, $F_{k_2}(F_{k_1}(\mathsf{H}(\cdot))) = F_{k_1}(F_{k_2}(\mathsf{H}(\cdot))) = \hat{F}_k(\mathsf{H}(\cdot))$ serves as a 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 the server and client respectively.

We then extend the above $(1, 1)$-RPMT protocol to $(n_1, 1)$-RPMT. Note that naive repetition by sending back $F_{k_2}(F_{k_1}(\mathsf{H}(y_i)))$ for each $y_i \in Y$ in the same order of the server's first move message $F_{k_1}(\mathsf{H}(y_i))$ does not lead to a secure $(n_1, 1)$-RPMT. This is because $\{\hat{F}_k(\mathsf{H}(y_i))\}_{i \in [n_1]}$ constitutes an order-preserving pseudorandom encoding of $(y_1, \dots, y_{n_1})$, and as a consequence, the server will learn the exact value of x if $x \in Y$. The idea to perform the membership test in an oblivious manner is making the pseudorandom encoding of $(y_1, \ldots, y_{n_1})$ independent of the order known by the server. A straightforward approach is to shuffle $\{\hat{F}_k(\mathsf{H}(y_i))\}$. This yields a $(n_1, 1)$-RPMT protocol from cwPRF, which can be batched to a full-fledged (n_1, n_2)-RPMT protocol by reusing the encoding key k_2. A simple calculation shows that for a (n_1, n_2)-RPMT protocol, the computation cost is $(n_1 + n_2)$ mappings, $(2n_1 + n_2)$ evaluations of F, n_2 lookups and one shuffling, and the communication cost is $(2n_1 + n_2)$ elements in the range of F. The resulting mqRPMT protocol is optimal in the sense that both computation and communication complexity are linear to the set size. To further reduce the communication cost, we can insert $\{\hat{F}(\mathsf{H}(y_i))\}$ into an order-hiding data structure such as a Bloom filter [Blo70] instead of shuffling them.

In Section 5.2, we show that cwPRF can be realized from DDH-like assumptions. Combining this with the above results, DDH implies all PSO protocols. Remarkably, it strikes back with an incredibly simple PSU protocol, once again demonstrating that the DDH assumption is truly a golden mine in cryptography.

mqRPMT from permuted OPRF. We choose $(n, 1)$-RPMT as the starting point of our second mqRPMT construction. The idea is *oblivious permuted encoding*, in which only one party say P_2 is able to encode, and the other party say P_1 learns the codewords of its elements $(y_1, \ldots, y_n)$ in a permuted order, while both parties learn nothing more. A tempting approach to implement this idea is using multi-point OPRF that underlies many PSI protocols [PRTY19, CM20]. More precisely, having P_1 (acts as the OPRF's client) and P_2 (acts as the OPRF's server) engage in an OPRF protocol. Eventually, P_1 obtains PRF values of $(y_1, \ldots, y_n)$ as encodings, and P_2 obtains a PRF key k. However, OPRF does not readily enable oblivious permuted encoding, because the standard OPRF functionality always gives the PRF values with the same order of inputs. To remedy this issue, we introduce a new cryptographic protocol called permuted OPRF (pOPRF). pOPRF can be viewed as a generalization of OPRF. The difference is that the server additionally obtains a random permutation π over $[n]$ besides PRF key k, while the client obtains PRF values in a permuted order as per π. pOPRF immediately implies a $(n, 1)$-RPMT protocol: The server P_1 with $Y = (y_1, \ldots, y_n)$ (acts as the pOPRF's client) and the client P_2 with $X = \{x\}$ (acts as the pOPRF's server) first engage in a pOPRF protocol. As a result, P_1 obtains $\{F_k(y_{\pi(i)})\}_{i \in [n]}$, and P_2 obtains a PRF key k and a permutation π over $[n]$. P_2 then computes and sends $F_k(x)$ to the server as an RPMT query for x. Finally, P_1 learns if $x \in Y$ by testing whether $F_k(x) \in \{F_k(y_{\pi(i)})\}_{i \in [n]}$, but learns nothing more since its received PRF values are of permuted order. At a high level, $F_k(\cdot)$ serves as an encoding function in mqRPMT-client's view, while $F_k(\pi(\cdot))$ serves as a permuted pseudorandom encoding function in mqRPMT-server's view. Extending the above $(n, 1)$-RPMT to full-fledged (n_1, n_2)-RPMT is straightforward by having the client reuse k and send $\{F_k(x_i)\}_{i \in [n_2]}$ as RPMT queries.

The question that remains is how to build pOPRF. One common approach to build OPRF is "mask-then-unmask". We choose this category of 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 different unmask procedure must be carried out accordingly. For this reason, 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, to recover outputs in permuted order. The above analysis indicates us that if the masking procedure can be done via a unifying manner, then the client 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 F_s to the intermediate input $\mathsf{H}(x)$, where H is a cryptographic hash function that maps input x to the domain of F_s. To enable efficient unmasking, we further require that F_s is a permutation and commutative with respect to F_k. This yields a simple pOPRF construction from commutative weak pseudorandom permutation. More precisely, to build pOPRF, the server picks a random PRP key k for F, while the client with input $X = (x_1, \ldots, x_n)$ picks a random PRP key s for F. The client then sends $\{F_s(\mathsf{H}(x_i))\}_{i \in [n]}$ to the server. Upon receiving the masked intermediate inputs, the server applies F_k to them, then sends the results in permuted order, a.k.a. $\{F_k(F_s(\mathsf{H}(x_{\pi(i)})))\}_{i \in [n]}$. Finally, the client applies F_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 due to the lack of nice algebra structures. This somehow explains the efficiency gap between the state-of-the-art PSI

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

and PCSI/PSU.

mqRPMT* from Sigma-mqPMT. To investigate the connection between mqRPMT and mqPMT, we first abstract a category of mqPMT protocols called Sigma-mqPMT, which is composed from 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 the algorithm $\mathsf{Test}(a, q, x, z)$. To facilitate efficient parallel composition, Sigma-PMT may satisfy the following three properties: (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; (iii) stateless testing property, which means the Test algorithm can be done without learning (q, x). By running multiple instances of Sigma-PMT with the above three properties in parallel, we obtain a three-move mqPMT protocol without increasing round complexity, called Sigma-mqPMT, which captures the common form of several PSI protocols [Mea86, FIPR05, CLR17]. By utilizing the stateless property, Sigma-mqPMT can be tweaked to mqRPMT* through the "shuffle-then-test" approach, without incurring computation and communication overhead. Therefore, a series of results in PSI setting can be translated 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 the unbalanced setting, which gives rise to the first PSU* protocol whose communication complexity is sublinear to the size of the large set X.

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

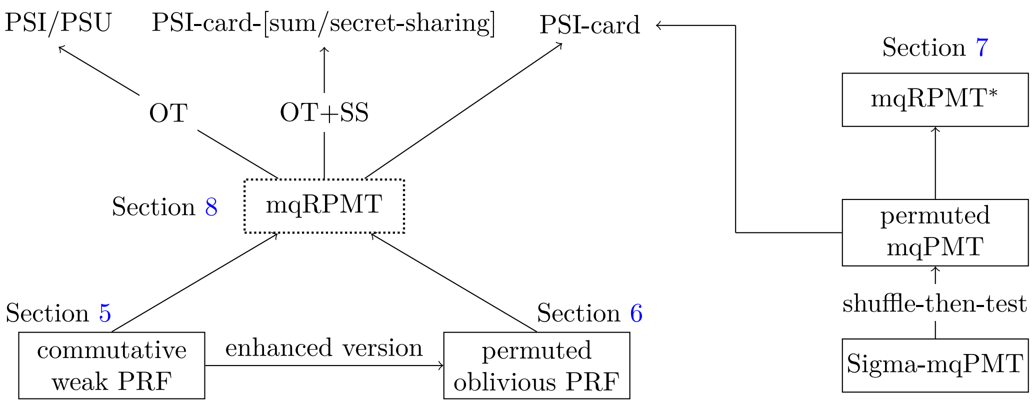


Figure 1: Summary of our main results. The rectangles with solid lines denote notions newly in this work. The rectangles with dotted lines denote notions from previous work.

1.4 Related Works

We review previous PSI-card, PSI-card-sum and PSU protocols that are relevant to our work. Ion et al. [IKN$^+$20] showed how to transform single-point OPRF-based [PSZ14, KKRT16], garbled Bloom filter-based [DCW13, RR17], and DDH-based [HFH99] PSI protocols into ones for computing PSI-card-sum by leveraging additively homomorphic encryption (AHE). However, their conversions are inefficient due to the usage of AHE, and as noted by the authors, detailed conversions to each category of protocols differ significantly, especially in the way of making use of the underlying AHE. In contrast, we distill Sigma-mqPMT from a broad class of PSI protocols, then show how to tweak it to mqRPMT* in a generic and black-box manner, without relying on any additional cryptographic tools. Our conversion works at a higher level, and this abstraction is useful for capturing technical ideas in a way that makes it more broadly applicable. Miao et al. [MPR$^+$20] put forward 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 to be easily built from commutative weak pseudorandom permutation 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]. Davidson and Cid [DC17] proposed a framework for constructing PSI, PSU, and PSI-card. Their protocols have linear complexity, but both the computation and communication complexity additionally rely on the statistical security parameter λ (a typical concrete choice is 40), resulting in low performance in practice. Kolesnikov et al. [KRTW19] showed that the performance of PSU in [DC17] is four orders of magnitude worse than the state-of-the-art at that time-being. Garimella et al. [GMR+21] proposed 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 all the enabling PSO protocols. Moreover, we note that the PSI-card-sum functionality defined in [GMR+21] differs 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], the party without holding values is given the cardinality and sum of the intersection. To distinguish this subtle difference, we refer to the functionality presented in [GMR+21] as reverse PSI-card-sum.

Concurrent work. Very recently, Zhang et al. [ZCL+23] proposed a generic construction of mqRPMT with linear complexity from oblivious key-value store, set-membership encryption and oblivious vector decryption-then-test protocol. 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, the computation complexity of their protocols has a large multiplicative constant (the statistical security parameter). Besides, as noted by the authors, their more efficient PKE-based construction is leaky, failing to meet the standard security of mqRPMT. Compared with their work, our construction of mqRPMT is much simpler. The instantiation meets the standard definition, and achieves strict linear complexity. Moreover, 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 $\{1, \ldots, n\}$, and use $\mathsf{Perm}[n]$ to denote all the permutations over 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, and its i-th 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 two-party protocol for computing the function $f(x_1, x_2)$, where party P_i has input x_i, and $\mathsf{output}(x_1, x_2)$ be the output of both parties in the protocol. For each party P_i where $i \in \{1, 2\}$, let $\mathrm{View}_{P_i}(x_1, x_2)$ denote the view of party P_i during an honest execution of Π on inputs x_1 and x_2, which consists of P_i's input, random tape, and all messages P_i received in the protocol.

Definition 2.1. Two-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 $i \in \{1, 2\}$:

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

Roughly speaking, a protocol is secure if P_i 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_1, and the receiver holds a set Y of size n_2 (we write $n_1 = n_2 = n$ in balanced setting). The ideal PSO functionality (depicted in Figure 2) computes the intersection, union, cardinality, intersection sum with cardinality and intersection secret-sharing with cardinality.

Parameters: The receiver P_1's input size n_1 and the sender P_2's input size n_2.

Inputs: The receiver P_1 inputs a set of elements $Y = \{y_1, \ldots, y_{n_1}\}$ where $y_i \in \{0,1\}^\ell$. The sender P_2 inputs a set of elements $X = \{x_1, \ldots, x_{n_2}\}$ where $x_i \in \{0,1\}^\ell$ and possibly a set of values $V = \{v_1, \ldots, v_{n_2}\}$ where $v_i \in \mathbb{Z}_p$ for some integer modular p.

Output:

- **intersection:** The receiver P_1 gets $X \cap Y$.
- **union:** The receiver P_1 gets $X \cup Y$.
- **union*****:** The receiver P_1 gets $X \cup Y$. The sender P_2 gets $|X \cap Y|$.
- **card:** The receiver P_1 gets $|X \cap Y|$.
- **card-sum:** The receiver P_1 gets $|X \cap Y|$. The sender P_2 gets $|X \cap Y|$ and $S = \sum_{i:x_i \in Y} v_i$.
- **card-secret-sharing:** The receiver P_1 gets $|X \cap Y|$ and $\{z_i^1\}_{i\in[n_2]}$. The sender P_2 gets $\{z_i^2\}_{i\in[n_2]}$. For each (z_i^1, z_i^2), $z_i^1 \oplus z_i^2 = x_i$ if $x_i \in Y$ and $z_i^1 \oplus z_i^2 = 0$ otherwise.

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

3 Protocol Building Blocks

3.1 Oblivious Transfer

Oblivious Transfer (OT) [Rab05] 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, the receiver learns m_b and neither party learns any additional information. In some cases, it suffices to use a "one-sided" version of OT, which conditionally transfers the only item of the sender or nothing to the receiver depending on the choice bit.

Though expensive public-key operations are 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 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 and message length ℓ.

Inputs: The sender P_1 inputs $\{(m_{i,0}, m_{i,1})\}_{i\in n}$, where each $m_{i,b} \in \{0,1\}^\ell$. The receiver P_2 inputs a bit vector $(b_1, \ldots, b_n) \in \{0,1\}^n$.

Output: The sender P_1 gets nothing. The receiver P_2 gets $\{m_{i,b_i}\}_{i\in[n]}$.

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

3.2 Multi-Query RPMT

mqRPMT [ZCL+23] is a protocol where the client with input vector $(x_1, \ldots, x_n)$ interacts with a server holding a set Y. As a result, the server learns only a bit vector $(e_1, \ldots, e_n)$ in which e_i indicates that whether $x_i \in Y$. Figure 4 formally defines the ideal functionality for mqRPMT. We also consider a relaxed version of mqRPMT called mqRPMT*, in which the client is also given $|X \cap Y|$.

Parameters: The server P_1's set size n_1 and number of RPMT queries n_2 by the client P_2.

Inputs: The server P_1 inputs a set $Y = (y_1, \ldots, y_{n_1})$, where $y_i \in \{0,1\}^\ell$. The client P_2 inputs a set $X = (x_1, \ldots, x_{n_2})$ (should be interpreted as a vector), where $x_i \in \{0,1\}^\ell$.

Output: The server P_1 gets a vector $\vec{e} = (e_1, \ldots, e_{n_2}) \in \{0,1\}^{n_2}$, where $e_i = 1$ if $x_i \in Y$ and $e_i = 0$ otherwise. The client P_2 gets nothing.

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

Family of PMT protocols. For completeness and fixing terminology, we explore the whole family of PMT protocols in a systematical way. We identify two characteristics of PMT protocols. One is direction, which consists of two options, namely forward or reverse. Forward option means the indication bits are finally known by the client, while reverse option means the indication bits are known by the server. The other one is order, which also consists of two options, namely ordered and permuted. The ordered option means the indication bits are of the right order known by the client. The permuted option means the indication bits are of the permuted order unknown to the client. By mixing-and-matching the 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+23]. 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 DH-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 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 $F : K \times D \to R$, 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 PRF is pseudorandomness.

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

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

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 $\mathcal{A}$. PRF that satisfy weak pseudorandomness are referred to as *weak PRF*.

4.1 Weak PRF from the DDH Assumption

We recall the weak PRF from the DDH assumption (implicitly presented in [NPR99]) as below.

- $\mathsf{Setup}(1^\kappa)$: runs $\mathsf{GroupGen}(1^\kappa) \to (\mathbb{G}, g, p)$, 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 $F_k(x) = x^k$.
- $\mathsf{KeyGen}(pp)$: 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 $(g^a, g^{b_1}, \dots, g^{b_n}, g^{ab_1}, \dots, g^{ab_n})$ and $(g^a, g^{b_1}, \dots, g^{b_n}, g^{c_1}, \dots, g^{c_n})$ 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 $(g^a, g^{b_1}, \dots, g^{b_n}, g^{c_1}, \dots, g^{c_n})$, $\mathcal{B}$ interacts with an adversary $\mathcal{A}$ in the weak pseudorandomness experiment, with the aim to determine if $c_i = ab_i$ or c_i is a random value.

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 with the knowledge of 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.

4.2 PRF from the DDH Assumption

Naor et al. [NPR99] showed how to convert a distributed weak PRF into a standard distributed PRF using random oracle heuristic. Actually, their approach can be generalized to bootstrap any weak PRF with dense domain to a standard PRF via the "hash-then-evaluate" formula. Concretely, to build a standard PRF with domain D from the DDH-based weak PRF described above, we first map the input element from D to $\mathbb{G}$ via a cryptographic hash function $\mathsf{H} : D \to \mathbb{G}$, then apply F_k in a cascade way, yielding a composite function $F_k \circ \mathsf{H} : D \to \mathbb{G}$. Assuming H is a random oracle, the pseudorandomness of the composite function $F_k \circ \mathsf{H}$ can be reduced to the weak pseudorandomness of F_k by leveraging the programmability of H. 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 $(g^a, g^{b_1}, \dots, g^{b_n}, g^{c_1}, \dots, g^{c_n})$, $\mathcal{B}$ interacts with an adversary $\mathcal{A}$ in the pseudorandomness experiment, with the aim to determine if $c_i = ab_i$ or c_i is a random value. $\mathcal{B}$ 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 (RO) 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 pseduorandomness of $F_k(\mathsf{H}(\cdot))$. □

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

5 The First Generic Construction of mqRPMT

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 $1/2$. 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. If F is a permutation, we say F is a commutative weak pseudorandom permutation (cwPRP).

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 PRFs 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 pair of cwPRF.

Remark 5.1. We note that our notion of 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.

5.3 mqRPMT from Commutative Weak PRF

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

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 $\hat{F}_k(\mathsf{H}(y_{\pi(i)})) \in \{\hat{F}_k(\mathsf{H}(x_i))\}_{i \in [n_2]}$.

Correctness. The above protocol is correct except the event 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. In what follows, we fix a tuple (x, y) such that $x \neq y$. Let E_0 be the event $\mathsf{H}(x) = \mathsf{H}(y)$. By the collision resistance of H, we have $\Pr[E_0] = 2^{-\kappa}$. Let E_1 be the event 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 weak pseudorandomness of F, we have $\Pr[E_{10}] = (1 - \Pr[E_0]) \cdot 1/|D|$, and $\Pr[E_{11}] = (1 - \Pr[E_0]) \cdot (1 - 1/|D|) \cdot 1/|D|$. If $|D| = \omega(\kappa)$, then both $\Pr[E_0]$, $\Pr[E_{10}]$ and $\Pr[E_{11}]$ are negligible in κ. Therefore, by union bound we have $\Pr[E] \leq n_1 n_2 \cdot (\Pr[E_0] + \Pr[E_{10}] + \Pr[E_{11}]) = \mathsf{negl}(\kappa)$.

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 $k_1 \xleftarrow{R} K$), and simulates with the knowledge of Y.

Parameters: The server P_1's set size n_1 and the client P_2's set size n_2, cwPRF $F : K \times D \to D$, and hash function $\mathsf{H} : \{0,1\}^\ell \to D$.

Inputs: The server P_1 inputs a set $Y = \{y_1, \dots, y_{n_1}\}$, where $y_i \in \{0,1\}^\ell$. The client P_2 inputs a set $X = \{x_1, \dots, x_{n_2}\}$ (should be interpreted as a vector), where $x_i \in \{0,1\}^\ell$.

Protocol:

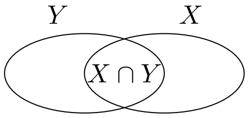
1. P_1 picks $k_1 \xleftarrow{\text{R}} K$, then computes and sends $\{F_{k_1}(\mathsf{H}(y_i))\}_{i \in [n_1]}$ to P_2.
2. P_2 picks $k_2 \xleftarrow{\text{R}} K$, then:
   (a) computes and sends $\{F_{k_2}(\mathsf{H}(x_i))\}_{i \in [n_2]}$ to P_1.
   (b) computes $\{F_{k_2}(F_{k_1}(\mathsf{H}(y_i)))\}_{i \in [n_1]}$, picks a random permutation π over $[n_1]$, then sends $\{F_{k_2}(F_{k_1}(\mathsf{H}(y_{\pi(i)})))\}_{i \in [n_1]}$ to P_1. An alternative choice instead of explicit shuffle is inserting $\{F_{k_2}(F_{k_1}(\mathsf{H}(y_i)))\}_{i \in [n_1]}$ 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 $\{F_{k_1}(F_{k_2}(\mathsf{H}(x_i)))\}_{i \in [n_2]}$, 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) $Y = (y_1, \dots, y_{n_1})$		P_2 (client) $X = (x_1, \dots, x_{n_2})$
$k_1 \xleftarrow{\text{R}} K$	$\xrightarrow{\{F_{k_1}(\mathsf{H}(y_i))\}_{i \in [n_1]}}$	
set $e_i = 1$ iff $F_{k_1}(F_{k_2}(\mathsf{H}(x_i))) \in \Omega$	$\xleftarrow{\{F_{k_2}(\mathsf{H}(x_i))\}_{i \in [n_2]}}$ $\Omega \leftarrow \{F_{k_2}(F_{k_1}(\mathsf{H}(y_{\pi(i)})))\}_{i \in [n_1]}$ $\Omega \leftarrow \mathsf{BF}(\{F_{k_2}(F_{k_1}(\mathsf{H}(y_{\pi(i)})))\}_{i \in [n_1]})$	$\pi \xleftarrow{\text{R}} \mathsf{Perm}[n_1]$ $k_2 \xleftarrow{\text{R}} K$

Figure 5: Multi-query RPMT from commutative weak PRF

- RO query: Sim_{P_2} emulates the random oracle H honestly. For each query $\langle z_i \rangle$, Sim_{P_2} picks $\alpha_i \xleftarrow{\mathrm{R}} D$, and assigns $\mathsf{H}(z_i) := \alpha_i$.
- Sim_{P_2} outputs $(F_{k_1}(\mathsf{H}(y_1)), \ldots, F_{k_1}(\mathsf{H}(y_{n_1})))$.



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

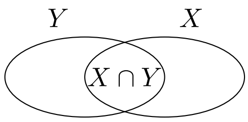
Clearly, Sim_{P_2}'s simulated view in Hybrid_1 is identical to P_2's real view.

Hybrid_2: Sim_{P_2} does not choose the randomness for P_1 (i.e., picks $k_1 \xleftarrow{\mathrm{R}} K$), 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 $(\eta_1, \ldots, \eta_{n_1})$ where $\eta_i \xleftarrow{\mathrm{R}} D$.

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 $\gamma_i \xleftarrow{\mathrm{R}} D$, 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 $\alpha_i \xleftarrow{\mathrm{R}} D$ and sets $\mathsf{H}(z_i) := \alpha_i$; if $z_i \in Y$, sets $\mathsf{H}(z_i) := \gamma_i$.
- outputs $(\eta_1, \ldots, \eta_{n_1})$.



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

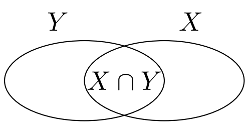
If $\eta_i = F_k(\gamma_i)$ for $i \in [n_1]$, then $\mathcal{A}$'s simulation is identical to Hybrid_1. If η_i are random values, then $\mathcal{A}$'s simulation is identical to Hybrid_2.

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 $(e_1, \ldots, e_{n_1})$, Sim_{P_1} chooses the randomness for P_2 (i.e., picks $k_2 \xleftarrow{\mathrm{R}} K$ and a random permutation π over $[n_1]$), 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 $\{F_{k_2}(\mathsf{H}(x_i))\}_{i \in [n_1]}$ and $\Omega \leftarrow \{F_{k_2}(F_{k_1}(\mathsf{H}(y_{\pi(i)}))\}_{i \in [n_1]}$.

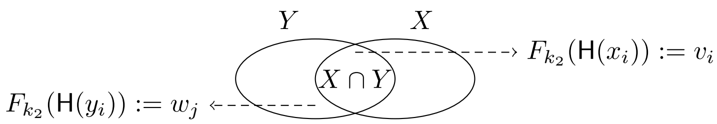


for $z_i \in D$, $\mathsf{H}(z_i) := \alpha_i \xleftarrow{\mathrm{R}} D$

Clearly, Sim_{P_1}'s simulation in Hybrid_1 is identical to the real view of P_1.

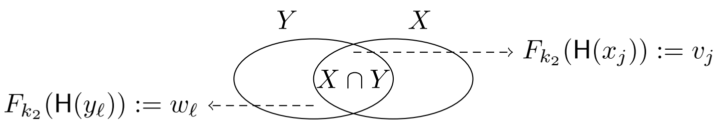
Hybrid_2: Sim_{P_1} does not choose randomness for P_2, and simulates without the knowledge of X. It simulates the random oracle H honestly as before, and changes its simulation of P_2's message. Let m be the Hamming weight of $(e_1, \ldots, e_{n_1})$.

- Sim_{P_1} picks $v_i \xleftarrow{\mathrm{R}} D$ for $i \in [n_2]$ (associated with $F_{k_2}(\mathsf{H}(x_i))$ where $x_i \in X$), outputs $\{v_i\}_{i \in [n_2]}$; picks $w_j \xleftarrow{\mathrm{R}} D$ for $j \in [n_1 - m]$ (associated with $F_{k_2}(\mathsf{H}(y_j))$ where $y_j \in Y - X \cap Y$), outputs a random permutation of $(\{F_{k_1}(v_i)\}_{e_i = 1}, \{F_{k_1}(w_j)\}_{j \in [n_1 - m]})$.



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 $n_1 + n_2 - m$ tuples (γ_i, η_i) where $\gamma_i \xleftarrow{\text{R}} D$, 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 $k_1 \xleftarrow{\text{R}} K$.

- RO queries: for $z_i \notin X \cup Y$, picks $\alpha_i \xleftarrow{\text{R}} D$ 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 $\{v_j\}_{j \in [n_2]}$; for each $z_i \in Y - X \cap Y$, $\mathcal{A}$ picks out the associated η_i to form $\{w_\ell\}_{\ell \in [n_1 - m]}$. Finally, $\mathcal{A}$ outputs $\{v_j\}_{j \in [n_2]}$ and a random permutation of $(\{F_{k_1}(v_j)\}_{x_j \in X \cap Y}, \{F_{k_1}(w_\ell)\}_{\ell \in [n_1 - m]})$.



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

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. □

Performance analysis. We now analyze the performance of the above (n_1, n_2)-mqRPMT protocol. Simple calculation shows that the total computation cost is $(n_1 + n_2)$ hashings, $(2n_1 + 2n_2)$ evaluations of cwPRF F, n_2 lookups whose complexity is $O(1)$, and one shuffling whose complexity is $O(n_1)$, while the total communication cost is $(2n_1 + n_2)$ elements in range D. In summary, both the computation and communication complexity are strictly linear in set sizes.

Optimization. The protocol can be further improved by inserting $\{F_{k_2}(F_{k_1}(\mathsf{H}(y_i)))\}_{i \in [n_1]}$ to a Bloom filter instead of explicitly shuffling them in the last move. In this way, the length of last message can be reduced from to n_1 group elements to $1.44\lambda \cdot n_1$ bits (with false positive probability $2^{-\lambda}$), where λ is the statistical security parameter and the typical choice is 40.

We highlight that our usage of Bloom filter is novel here since it crucially explores the order-hiding property[3]. To the best of knowledge, Bloom filter merely serves as a space-efficient data structure in previous works [KLS+17, RA18].

6 The Second Generic Construction of mqRPMT

6.1 Definition of Permuted OPRF

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

We consider an extension of OPRF which we called permuted OPRF (pOPRF). Roughly speaking, the server additionally picks a random permutation π over $[n]$, and the client 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.

[3]Formally, order-hiding property means that the data structure does not reveal the adding order of elements. Recall that an empty Bloom filter is a bit array of m bits (all set to 0), and adding an element x is done by setting the bits at positions $\{h_1(x), \ldots, h_k(x)\}$ to be 1, where $\{h_i\}_{i=1}^k$ are k distinct hash functions. Clearly, Bloom filter satisfies order-hiding property since the resulting Bloom filter is independent of the adding order. We also stress that the choice of Bloom filter is not arbitrary here, cause other filters such as Cuckoo filter and Vacuum filter do no meet order-hiding property.

Parameters: Number of OPRF queries n.

Inputs: The server P_1 inputs nothing. The client P_2 inputs a set $X = (x_1, \ldots, x_n)$, where $x_i \in \{0,1\}^\ell$.

Output: The server P_1 gets a random PRF key k and a random permutation π over $[n]$. The client P_2 gets $y_i = F_k(x_{\pi(i)})$.

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 cwPRP F. At a high level, the unified masking procedure is done by applying a weak PRF $F_s(\cdot)$ to $\mathsf{H}(x)$, and the unmasking process is enabled by the commutative property of F and the fact that $F_s(\cdot)$ is an efficiently invertible permutation. We depict the construction in Figure 7.

$$F : K \times D \to D,\ \mathsf{H} : \{0,1\}^\ell \to D$$

P_1 (sender)		P_2 (receiver) $X = (x_1, \ldots, x_n)$
	$\xleftarrow{\{F_s(\mathsf{H}(x_i))\}_{i \in [n]}}$	$s \xleftarrow{R} K$
$k \xleftarrow{R} K,\ \pi \xleftarrow{R} \mathsf{Perm}[n]$	$\xrightarrow{\{F_k(F_s(\mathsf{H}(x_{\pi(i)})))\}_{i \in [n]}}$	$F_k(\mathsf{H}(x_{\pi(i)})) \leftarrow F_s^{-1}(F_k(F_s(\mathsf{H}(x_{\pi(i)}))))$

Figure 7: Permuted OPRF from cwPRP

Remark 6.1. We note that it suffices to build permuted OPRF from a tuple of cwPRF (F_k, G_s) where G_s is a weak permutation.

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 and* F *is a family of cwPRPs.*

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

- 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 $(F_s(\beta_1), \ldots, F_s(\beta_n))$, where $\mathsf{H}(x_i) = \beta_i$.

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 F_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 $(\eta_1, \ldots, \eta_n)$.

Clearly, if $\eta_i = F_s(\gamma_i)$, $\mathcal{A}$ 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 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 $X = (x_1, \ldots, x_n)$ 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 $\{F_s(F_k(\mathsf{H}(x_{\pi(i)})))\}_{i\in[n]}$.

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 is actually a family of cwPRPs. 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 cwPRP, 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.

Parameters: hash function $\mathsf{H} : \{0,1\}^\ell \to \mathbb{G}$.

Inputs: The server P_1 inputs nothing. The client P_2 inputs a set $X = \{x_1, \ldots, x_n\}$, where $x_i \in \{0,1\}^\ell$.

Protocol:

1. P_2 picks $s \xleftarrow{R} \mathbb{Z}_p$, then sends $(\mathsf{H}(x_1)^s, \ldots, \mathsf{H}(x_n)^s)$ to the sender P_1.
2. P_1 picks $k \xleftarrow{R} \mathbb{Z}_p$ 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 based on the DDH assumption

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 using a collision resistant hash function.

Correctness. The above protocol is correct except the case $E = \vee_{i,j} E_{ij}$ occurs, 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 $\Pr[E] \leq n_1 n_2 \cdot \Pr[E_{ij}] = n_1 n_2 / 2^\ell = \mathsf{negl}(\lambda)$.

Parameters: The server P_1's set size n_1 and the client P_2's set size n_2, a permuted OPRF for $F : K \times D \to R$.

Inputs: The server P_1 inputs a set $Y = \{y_1, \dots, y_{n_1}\}$, where $y_i \in \{0,1\}^\ell$. The client P_2 inputs a set $X = \{x_1, \dots, x_{n_2}\}$, where $x_i \in \{0,1\}^\ell$.

Protocol:

1. P_1 with inputs $Y = \{y_1, \dots, y_{n_1}\}$ and P_2 engage in a permuted OPRF protocol. P_1 acts as the pOPRF's client, while P_2 acts as the pOPRF's server. At the end of the protocol, P_1 obtains $\{F_k(y_{\pi(i)})\}_{i\in[n_1]}$, P_2 obtains a PRF key k and a random permutation π over $[n_1]$.
2. P_2 computes and sends $(F_k(x_1), \dots, F_k(x_{n_1}))$ to P_1.
3. P_1 sets $e_i = 1$ iff $F_k(x_i) \in \{F_k(y_{\pi(i)})\}_{i\in[n_1]}$.

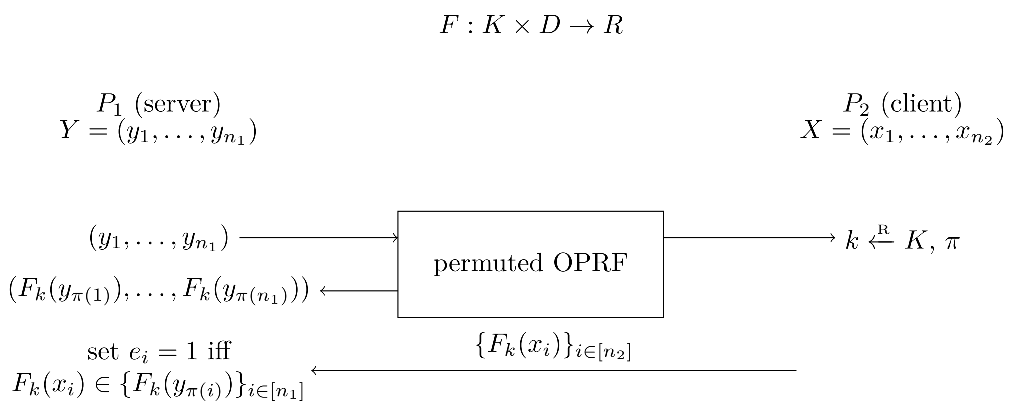


Figure 9: mqRPMT from permuted OPRF

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

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.

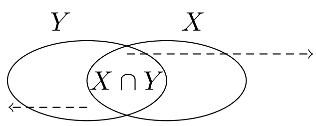
Hybrid$_1$: Given P_1's input $Y = (y_1, \ldots, y_{n_1})$ and output $(e_1, \ldots, e_{n_2})$, Sim_{P_1} creates the simulated view as below:

- pick a random PRF key k and a random permutation π over $[n_1]$;
- compute $(F_k(y_{\pi(1)}), \ldots, F_k(y_{\pi(n_1)}))$, then generate its stage 1's view by invoking the simulator for P_1 of permuted OPRF with input $(y_1, \ldots, y_{n_1})$ and output $(F_k(y_{\pi(1)}), \ldots, F_k(y_{\pi(n_1)}))$;
- generate stage 2's view $(F_k(x_1), \ldots, F_k(x_{n_2}))$ using k with the knowledge of P_2's input X.

The simulated stage 2's view is identical to the real one. By the semi-honest security of permuted OPRF on P_1' side, the stage 1's simulated view is computationally indistinguishable to 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 $(\eta_1, \ldots, \eta_{n_2})$, where $\eta_i \xleftarrow{\text{R}} R$; 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 $\{w_\ell\}_{\ell \in [n_1 - m]}$; apply a random permutation Π of $(\{v_j\}_{j \in [m]}, \{w_\ell\}_{\ell \in [n_1 - m]})$, treat the result as $(F_k(y_{\pi(1)}), \ldots, F_k(y_{\pi(n_1)}))$ (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 $(y_1, \ldots, y_{n_1})$ and output $(F_k(y_{\pi(1)}), \ldots, F_k(y_{\pi(n_1)}))$.

$\{F_k(y_{\pi(i)})\}_{i \in [n_1]} := \Pi(\{v_j\}_{j \in [m]}, \{w_\ell\}_{\ell \in [n_1 - m]})$ 

$F_k(x_i) := \eta_i$

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 $(x_1, \ldots, x_{n_1})$, output the responses $(\eta_1, \ldots, \eta_{n_1})$.
- pick a random permutation π over $[n_1]$;
- query the real-or-random oracle with $(y_{\pi(1)}, \ldots, y_{\pi(n_1)})$ and obtain $(\zeta_1, \ldots, \zeta_{n_1})$ in return; then generate its stage 1's view by invoking the simulator for P_1 of permuted OPRF with input $(y_1, \ldots, y_{n_1})$ and output $(\zeta_1, \ldots, \zeta_{n_1})$.

Clearly, if $\mathcal{A}$ queries the real oracle, then its simulation is identical to 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. □

Performance analysis. The performance of the above (n_1, n_2)-mqRPMT protocol is dominated by the underlying permuted OPRF protocol. We analyze the efficiency of the concrete (n_1, n_2)-mqRPMT protocol obtained from the DDH-based permuted OPRF (shown in Figure 8). Simple calculation shows that the total computation cost is $(n_1 + n_2)$ hashings, $3n_1$ scalar multiplications, n_2 scalar inverse muiltiplications, n_2 lookups whose complexity is $O(1)$, and one shuffling whose complexity is $O(n_1)$, while the total communication cost is $(2n_1 + n_2)$ group elements in $\mathbb{G}$.

Comparison of the two constructions of mqRPMT. We have presented two generic constructions of mqRPMT, the first is from cwPRF, while the second is from pOPRF. We summarize their differences as below.

- cwPRF-based construction has a practical advantage since it admits pretty fast implementation based on `Curve25519` (causing only scalar multiplication needed) as well as the Bloom filter trick to reduce communication cost.
- Compared to the cwPRF-based construction, the pOPRF-based construction does not admit fast implementation based on `Curve25519` since scalar inverse multiplication operation like $g^{s^{-1}}$ is not supported due to automatic scalar clamp (see Section 9 for technical details), and the Bloom filter optimization is not applicable. Nevertheless, the pOPRF-based construction can be viewed as a counterpart of OPRF-based mqPMT construction, and thus is more of theoretical interest. So far, we only know how to build pOPRF based on assumptions with nice algebra structure, but not from efficient symmetric-key primitives. This somehow explains the efficiency gap between mqPMT and mqRPMT.

7 Connection Between mqRPMT and 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 the following pattern.

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

After receiving z, the 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 a, 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.

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

Figure 10: Sigma-mqPMT

Looking ahead, to reduce the semi-honest security of mqRPMT* to that of Sigma-mqPMT, we assume the simulator $\mathsf{Sim}(X, \vec{e})$ for the client P_2 in mqRPMT* 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 mqRPMT* from 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 send 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.

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:

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

Figure 11: mqRPMT* from Sigma-mqPMT

- for $1 \leq i \leq n_2$, run $\mathsf{Sim}'(e^*_i; r_i) \to z^*_i$, obtaining $\vec{z}^* = (z^*_1, \dots, z^*_{n_2})$.
- 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_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: 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, given $(Y, \vec{e})$ it first invokes the Sigma-mqPMT's simulator for P_1 on input Y to generate $\vec{q}$, then picks a random permutation π over $[n_2]$ and 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. From 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

8.1 PSO Framework from mqRPMT

We show how to build a PSO framework centering around mqRPMT in Figure 12.

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 $|X \cap Y| = m$.

Parameters: The receiver P_1's set size n_1 and the client P_2's set size n_2.

Inputs: The receiver P_1 inputs a set $Y = \{y_1, \ldots, y_{n_1}\}$, where $y_i \in \{0,1\}^\ell$. The sender P_2 inputs a set $X = \{x_1, \ldots, x_{n_2}\}$ and $V = \{v_1, \ldots, v_{n_2}\}$, where $x_i \in \{0,1\}^\ell$ and $v_i \in \mathbb{Z}_p$. Let q be a big integer greater than $n_2 \cdot p$.

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_1 holding e_i and P_2 holding $(\perp, x_i)$ invoke one-sided OT n_2 times. P_1 learns $\{x_i \mid e_i = 1\}_{i \in [n_2]} = X \cap Y$.
   - **union:** P_1 holding e_i and P_2 holding $(x_i, \perp)$ invoke one-sided OT n_2 times. P_1 learns $\{x_i \mid e_i = 0\}_{i \in [n_2]} = X \backslash Y$, and outputs $\{X \backslash Y\} \cup Y = X \cup Y$.
   - **card-sum:** P_2 randomly picks $r_i \in \mathbb{Z}_q$ and computes $r' = \sum_{i=1}^{n_2} r_i \bmod q$. Subsequently, P_1 holding e_i and P_2 holding $(r_i, r_i + v_i)$ invoke 1-out-of-2 OT n_2 times. P_1 learns $S' = \sum_{i=1}^{n_2} r_i + e_i \cdot v_i \bmod q$, then sends S' and the Hamming weight of $\vec{e}$ to P_2. P_2 computes $S = (S' - r') \bmod q$.
   - **card-secret-sharing:** P_2 randomly picks $r_i \in \{0,1\}^\ell$. Subsequently, P_1 holding e_i and P_2 holding $(r_i, r_i \oplus x_i)$ invoke 1-out-of-2 OT n_2 times. P_1 learns $\{z_i\}_{i \in [n_2]}$, and thus $\{(z_i, r_i \oplus x_i)\}_{e_i = 1}$ constitutes the shares of intersection values.

Figure 12: PSO from mqRPMT

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 the client side and the semi-honest security for OT on the 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 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 the server side and the semi-honest security for OT on the receiver side, the simulation is indistinguishable to the real view via standard hybrid argument.

This proves the theorem. $\square$

We compare our PSI-card protocol with closely related protocols. Huberman et al. [HFH99] proposed the first PSI-card protocol but did not provided security proof. Agrawal et al. [AES03] explained and proved the classic protocol via the notion of "commutative encryption". Later, De Cristofaro et al. [CGT12] gave a close variant of the classic protocol. Our PSI-card protocol is generically derived from the more abstract mqRPMT, which in turn can be built from cwPRF or pOPRF. By instantiating the underlying cwPRF and pOPRF from the DDH assumption, we recover the PSI-card protocols in [HFH99, CGT12] respectively. To sum up, the merit our generic PSI-card construction from mqRPMT is that it not only encompasses existing concrete protocols, but also readily profits from the possible improvements of the underlying mqRPMT (e.g., Bloom filter optimization and post-quantum secure realization based on the EGA assumption).

We 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 protocols presented in [IKN+20] are 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, mqRPMT underlying our protocol is conceptually simpler than its permuted version. More importantly, mqRPMT admits instantiations with optimal linear complexity, while the current best instantiation of permuted mqRPMT requires superlinear 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 do 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 send back the masked sum value to the receiver, and the receiver finally recovers the intersection sum by unmasking.

We also briefly discuss the differences between our card-secret-sharing protocol with related work. The most related functionality is circuit-PSI [HEK12, PSTY19, RS21]. The only difference between our card-secret-sharing and circuit-PSI is that our protocol additionally leaks the cardinality to the receiver. However, as pointed out by Garimella et al. [GMR+21], in many applications of interest, the functions that need to be computed indeed imply such leakage. Garimella et al. [GMR+21] also proposed a similar functionality named secret-shared intersection, in which the parties only get the sharing of intersection elements. As a result, their protocol leaks the cardinality to both the sender and the receiver.

Malicious security. Our PSO framework is secure against semi-honest adversaries. To attain malicious security, there are two main challenges. First, the underlying mqRPMT and OT must be secure in the malicious setting. Second, even if both mqRPMT and OT satisfy malicious security, the resulting PSO protocols still fail to satisfy malicious security, because a malicious adversary may cause the output of mqRPMT inconsistent with the input of OT. For example, in the PSU protocol, regardless of the output of mqRPMT, a malicious adversary could set the selection bits for all OT instances to be '0', and thus obtains all the elements of the sender. To boost mqRPMT from semi-honest security to malicious security and ensure output-input consistency, it seems that there is no better way than using zero-knowledge proofs, which would greatly affect the efficiency. We left efficient PSO framework with malicious security as an interesting open problem.

8.2 Private-ID

Recently, Buddhavarapu et al. [BKM+20] proposed 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 with 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 any desired private computation item by item, being assured that identical items are aligned. Buddhavarapu et al. [BKM+20] also gave a concrete DDH-based private-ID protocol. Garimella et al. [GMR+21] showed 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 roles, 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, 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 over the input distribution. Such relaxation may lead to remarkable efficiency improvement.

In this work, we instantiate the generic private-ID construction as below: (1) realize the distributed OPRF protocol by running the current most efficient multi-point OPRF of [RR22] twice in reverse order; (2) run the PSU protocol from cwPRF-based mqRPMT with the obtained two sets of pseudorandom identifiers as inputs to fulfill the private-ID functionality.

Parameters: PRF $G : K \times D \to R$, where $K = K_1 \times K_2$.

Inputs: P_1 inputs a set $X = \{x_1, \dots, x_{n_1}\}$, where $x_i \in D$. P_2 inputs a set $Y = \{y_1, \dots, y_{n_2}\}$, where $y_i \in D$.

Output: P_1 gets $\{G_{k_1,k_2}(x_i)\}_{i \in [n_1]}$ and k_1. P_2 gets $\{G_{k_1,k_2}(y_i)\}_{i \in [n_2]}$ 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 a weaker security notion by allowing the real-ideal indistinguishability to 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. Observe that the output of distributed DDH-based OPRF are already pseudorandom EC points. In this case, it suffices to use distributional DDH-based PSU instead, and thus can save $2n$ hash-to-point operations, which are costly in the 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) **psi-card**: cardinality of the intersection; (3) **psi-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, our implementation is organized in a modular and unified fashion: first implement the core mqRPMT protocol, then build various PSO protocols upon it. Besides, it only requires OpenSSL [Opea] as the main 3rd party library, and can smoothly run on both Linux and x86_64 MacOS platforms.

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 cores) 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 `iptables` command to calculate the communication cost, 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 test the balanced scenario by setting the input set size $n_1 = n_2$ (our implementation supports unbalanced scenario as well), and randomly generate two input sets with 128 bits item length conditioned on the intersection size being roughly $0.5n$. The exception is the protocol in [GMR+21], whose item length is set as 61 bits in default and cannot exceed 64 bits.
- The PSI-card-sum protocol [IKN+20] and the private-id protocol [BKM+20] are two of the related works we are going to compare. The former implementation is built upon `NIST P-256` (also known as `secp256r1` and `prime256v1`), while the latter implementation is built upon `Curve25519`. For a fair and comprehensive comparison, we implement our protocols under both standard elliptic curve `NIST P-256` and special elliptic curve `Curve25519`. For protocols based on `NIST P-256`, we denote the one not using or using point compression technique with ⧫ and ▼ respectively. For protocols based on `Curve25519`, we denote with ★.

9.3 Evaluation of mqRPMT

We first report the performance of our cwPRF-based mqRPMT protocol (optimized with Bloom filter) described in Section 5.3, which dominates the communication and computation overheads of its enabling PSO protocols. We test our protocol up to 4 threads, since both the server and the client run on a single

CPU with 8 physical cores. Our cwPRF-based mqRPMT achieves optimal linear complexity, and thus is scalable, which is demonstrated by the experimental results in Table 2. Moreover, the computation tasks on both sides in our cwPRF-based mqRPMT are highly parallelable, thus we can effortlessly using OpenMP [Opeb] to make the program multi-threaded.

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$^\blacklozenge$	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$^\blacktriangledown$	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 ×	–	–	–
mqRPMT$^\bigstar$	1	0.26	3.51	54.85	0.81	5.41	68.68	0.26	4.23	67.66
	2	0.15	1.79	28.24	0.75	3.83	41.38			
	4	0.10	1.07	15.32	0.72	3.09	28.31			
Speedup		1.7-2.6 ×	2.0-3.3 ×	1.9-3.6 ×	1.1-1.1 ×	1.4-1.8 ×	1.7-2.4 ×	–	–	–

9.4 Benchmark Comparison of PSO Protocols

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 DH-PSI protocol reported in [PRTY19] and re-implemented by ourselves. We remark that the PSI protocols in comparison are not competitive to the state-of-the-art PSI protocol. We include them merely for illustrative purpose and completeness. PSI protocols from public-key techniques are used to be thought inefficient, but our experiment results demonstrate that they could be practical by using modern crypto library and carefully choosing optimized parameters. By leveraging fast elliptic curve operations provided by OpenSSL, our mqRPMT-based PSI protocol is $3.4-10.5\times$ faster than the DH-PSI protocol[4] implemented in [PRTY19]. By further exploiting the features of `Curve25519` in important ways (see Section 9.5 in details), our re-implemented DH-PSI protocol (denoted by DH-PSI$^\bigstar$) achieves a $6.3 - 26.1\times$ speedup, which is arguably the most efficient implementation known to date.

[4] We remark that except inefficiency, their implementation also has a severe security issue. More precisely, they realize the hash-to-point function $\{0,1\}^* \to \mathbb{G}$ as $x \mapsto g^{\mathsf{H}(x)}$, where H is some cryptographic hash function. However, such hash-to-point function cannot be modeled as random oracle anymore since it exposes the algebra structure of output in the clear, and hence totally compromise security. Similar issue also appears in `libPSI`.

Table 3: Communication cost and running time of PSI protocol.

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]$^{\bigstar}$	5.51	88.64	1418.20	5.82	90.79	1498.67	0.30	4.74	76.60
Our PSI$^{\blacklozenge}$	**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**
Our PSI$^{\bigstar}$	**0.29**	**3.56**	**55.11**	**1.19**	**6.38**	**75.56**	**0.40**	**6.25**	**99.71**
DH-PSI$^{\bigstar}$	**0.22**	**3.39**	**54.79**	**0.92**	**5.57**	**69.31**	**0.28**	**4.57**	**74.1**

Recently, Rosulek and Trieu [RT21] proposed a PSI protocol based on Diffie-Hellman key agreement, which requires the least time and communication of any known PSI protocols for small sets. Somewhat surprisingly, Table 4 shows that for small sets our mqRPMT-based PSI protocol is faster than their protocol in the LAN setting, and our re-implemented DH-PSI is faster than their protocol in all settings with marginally larger communication cost.

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

PSI	Running time (ms)						Comm. (KB)		
	LAN			WAN			total		
	2^{8}	2^{9}	2^{10}	2^{8}	2^{9}	2^{10}	2^{8}	2^{9}	2^{10}
[RT21]$^{\bigstar}$	50.0	71.0	147.3	224.1	260.2	457.9	17.9	34.1	66.3
Our PSI$^{\bigstar}$	**41.9**	**69.5**	**99.3**	**577.0**	**582.9**	**646.1**	**38.6**	**63.5**	**113.3**
DH-PSI$^{\bigstar}$	**16.49**	**31.80**	**56.91**	**210.42**	**227.33**	**252.32**	**18.48**	**36.68**	**72.8**

PSI-card. We compare our mqRPMT-based PSI-card protocol to the PSI-card protocol in [GMR$^+$21]. Table 5 shows that our protocol achieves a $2.4 - 10.5\times$ speedup, and reduces the communication cost by a factor of $10.9 - 14.8\times$.

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

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$^+$21]	1.00	8.41	126.01	8.60	27.46	323.52	2.93	55.49	1030
Our PSI-card$^{\blacklozenge}$	**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**
Our PSI-card$^{\bigstar}$	**0.27**	**3.51**	**54.89**	**0.82**	**5.42**	**68.31**	**0.26**	**4.23**	**67.70**

Remark 9.1. It is interesting to examine if one can leverage the state-of-the-art circuit PSI [RR22] to build an efficient PSI-card. In circuit PSI, the sender with X and the receiver with Y obtain shares of an indication bit string e of length m, where $e_{\pi(i)} = 1$ if and only if $y_i \in X \cap Y$, and π is an injection known to the receiver, indicating the position of the i-th element y_i inserted by the receiver in the cuckoo hash table. To construct PSI-card from circuit PSI, a simple idea is to have one party send his shares of e to the other party on the first place, then the other party can reconstruct e and compute its Hamming weight. However, this method is not secure. If the receiver sends shares to the sender, the sender learns the positions of the receiver's elements in the cuckoo hash table, which may reveal information about receiver's input set Y; if the sender sends shares to the receiver, the receiver directly obtain the intersection since it knows the corresponding elements for each bit of e. In light of the above reasoning, general 2PC technique seems unavoidable, in which the circuit under computation has to convert boolean inputs to arithmetic one and sum the result. It is unclear whether the size of such circuit is small. For this reason, in this work we do not compare our PSI-card protocol to the one from circuit PSI.

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 more 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, compared with the protocol presented in [IKN+20], our protocol achieves a $28.5 - 76.3\times$ improvement in running time and a $7.4\times$ improvement in communication cost.

Table 6: Communication cost and running time of PSI-card-sum protocol.

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]$^\blacktriangledown$ (deployed)	23.64	176.34	–	30.10	186.29	–	2.72	43.24	–
Our PSI-card-sum$^\blacklozenge$	**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**
Our PSI-card-sum$^\bigstar$	**0.31**	**3.73**	**57.44**	**1.36**	**6.53**	**76.16**	**0.37**	**5.75**	**91.70**

We assume each associated value is a non-negative integer in $[0, 2^{32})$ conditioned on the upper bound of intersection sum being 2^{32}. We note that the implementation of [IKN+20] only works in our environment at set sizes 2^{12} and 2^{16}. For set size 2^{20}, we encounter a run time error reported in [Pri] that has not been fixed yet. The corresponding cells are marked with "–".

PSU. We compare our mqRPMT-based PSU protocol to the state-of-the-art PSU protocols in [GMR+21, ZCL+23, JSZ+22]. The work [ZCL+23] provides two PSU protocols from public-key and symmetric-key respectively. The work [JSZ+22] also provides two PSU protocols called PSU-S and PSU-R. We choose the most efficient PKE-PSU [ZCL+23] and PSU-R [JSZ+22] for comparison. Among all the mentioned PSU protocols, only our PSU protocol achieves strict linear communication and computation complexity. The experimental results in Table 7 indicate that our PSU protocol is the most superior one. Comparing to the state-of-the-art PSU protocol of [ZCL+23], our protocol achieves a $2.7 - 17\times$ improvement in running time and a $2\times$ improvement in communication cost.

Table 7: Communication cost and running time of PSU protocol.

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+23]$^\blacklozenge$	4.87	12.19	141.38	5.78	15.75	182.88	1.35	21.41	342.38
[ZCL+23]$^\blacktriangledown$	5.10	15.13	187.29	5.82	17.37	210.06	0.77	12.20	195.17
[JSZ+22]	2.29	8.50	516.04	5.33	27.00	736.30	3.59	70.37	1341.55
Our PSU$^\blacklozenge$	**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**
Our PSU$^\bigstar$	**0.30**	**3.55**	**55.48**	**1.19**	**6.38**	**74.96**	**0.40**	**6.25**	**99.71**

Private-ID. We compare our concrete private-ID protocol described in Section 8.2 to the state-of-the-art protocols in [BKM+20, GMR+21]. As shown in Table 8, our private-ID protocol achieves a $1.9 - 5.9\times$ speedup comparing to the current most computation efficient private-ID protocol [GMR+21], while requires $1.3\times$ less communication for sufficiently large sets[5] than the current most communication

[5] We note that our protocol requires more communication for sets of size 2^{12}. This is because the underlying OPRF is built from OKVS and PCG-style VOLE [RR22], whose computation and communication complexities have relatively large constant terms, leading to noticeable fixed costs.

efficient private-ID protocol [BKM+20]. Hence, our protocol is arguably the most computation and communication efficient private-ID to date.

Table 8: Communication cost and running time of private-ID protocol.

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$^\blacklozenge$	**0.50**	**7.26**	**116.26**	**7.68**	**18.17**	**167.09**	**2.80**	**16.46**	**233.76**
Our Private-ID$^\blacktriangledown$	**0.57**	**8.36**	**133.33**	**7.73**	**18.38**	**173.90**	**2.55**	**12.45**	**169.74**
Our Private-ID*	**0.28**	**3.63**	**58.02**	**7.28**	**13.64**	**97.66**	**2.54**	**12.32**	**167.70**

9.5 Tips For ECC-based Implementations

In what follows, we summarize the lessons we learned during the implementation of ECC-based protocols, with the hope to uncover some dark details and correct imprecise impressions.

We first highlight the following two caveats when implementing with standard elliptic curves:

- *Pros and cons of point compression technique.* Point compression is a standard trick 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 scalar multiplication. Our perspective is that point compression offers a natural trade-offs between communication and computation. The above experimental results demonstrate that the total running time gives a large weight to communication cost in bandwidth constrained scenarios. Therefore, in the WAN setting (involving parties cannot be co-located) we recommend not to apply point compression trick, while in the LAN setting (involving parties are co-located) we recommend to apply point compression trick. A quick take-away is that point compression trick pays off in the setting where communication is much more expensive than computation.

- *Tricky hash-to-point operation.* The hash to point operation is very tricky in ECC. So far, there is no universal method to securely map arbitrary bit strings to points on elliptic curves. Here, the vague term "securely" 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, even such simple hash-to-point operation could be as expensive as scalar multiplication, which should be avoided if possible.

Regarding the two caveats discussed above, the following questions arise: (1) is it possible to get the best of two worlds of point compression; (2) could the hash-to-point operation be cheaper. Luckily, the answers are affirmative under some circumstances.

With the aim to avoid many potential implementation pitfalls, Bernstein [Ber06] designed an elliptic curve dedicated to ECDH function known as `Curve25519` in 2005. Due to its many efficiency/security advantages, it has been widely deployed in numerous applications and has become the *de facto* alternative to `NIST P-256`. Here, we highlight its two nice features that are particularly beneficial for our cwPRF-based mqRPMT protocol: (i) it allows efficient scalar multiplication in compressed form (only X coordinates); (ii) by design, any 32-byte bit string (interpreting as X coordinate) can be ambiguously identified as a valid point on curve. Feature (i) brings us the best of two worlds of point compression, without making trade-off anymore, while feature (ii) makes the hash-to-point operation almost free, simply hashing the input to a 32-byte bit string via cryptographic hash function. Naturally, `Curve25519` has deficiencies coming with its nice features. All the known implementations of `Curve25519` that support efficient scalar multiplication in X-coordinate compressed form do not provide interfaces for point addition, subtraction, and scalar inverse multiplication. The reason is that (a) point addition and subtraction operations cannot be performed using only X coordinates, thus in turn requiring expensive decompression operation; (b) giving any 32-byte integer value as the scalar, existing implementations would automatically "clamp" it before scalar multiplication, thus requiring complicated treatment to support scalar inverse multiplication.

Luckily, our cwPRF-based mqRPMT protocol only requires scalar multiplication and hash-to-point operations, and thus can enjoy the nice features without being affected by the deficiencies. This explains the advantages of our cwPRF-based mqRPMT protocol based on `Curve25519` over that based on `NIST P-256`. To the best of our knowledge, this is the first time that `Curve25519` fully unleashes its advantages in the area of private set operations. Prior to this work, Rosulek and Tireu [RT21] employed `Curve25519` to build a PSI protocol from Diffie-Hellman key agreement (DHKA) with strongly uniform property [FMV19], whose instantiation inherently requires the elligator encoding/decoding mechanism [BHKL13]. Henceforth, the optimizations originated from feature (i) and (ii) do not apply to their construction because it requires encoding/decoding EC points to bit strings (thus the EC points cannot only be represented by X coordinates), rather than hashing elements to EC points. In summary, for protocols that are not involved with point addition/subtraction and scalar inverse multiplication, `Curve25519` would be a good choice.

Public-key operations are always rashly thought to be much expensive than symmetric-key operations, and thus the design philosophy of many practical protocols opts to avoid public-key operations. Our experimental results indicate that this impression is not precise anymore after rapid advances on ECC-based cryptography in recent years. By leveraging optimized implementation, public-key operations could be as efficient as symmetric-key operations. As a concrete example, in EC group with 128 bit security level one EC point scalar operation takes 0.026 ms and one EC point addition takes 0.00028 ms on a laptop.

10 Summary and Perspective

This work demonstrates that mqRPMT protocol is complete for most private set operations. In particular, we created a unified PSO framework from mqRPMT, which is rather attractive given its conceptual simplicity and modularity. The high level abstraction is useful for allowing us to interpret various PSO protocols through the lens of mqRPMT, and helps to greatly reduces the deployment and maintenance costs of PSO in the real world. We also presented two generic constructions of mqRPMT and instantiated them from the DDH assumption, yielding a family of PSO protocols with optimal asymptotic complexity and good concrete efficiency that are superior and competitive to existing ones. To sum up, we regard the PSO framework from mqRPMT together with its efficient implementations as the main contribution of this work. We emphasize that our framework does not intend to fully cover the current state of the art, which is a rapidly moving target. Instead, it mainly aims to suggest common principles and clean abstractions that can apply broadly and systematically.

Along the way of constructing mqRPMT, we introduce cwPRF and pOPRF. The notion cwPRF can be viewed as the right cryptographic abstraction of the celebrated DH functions, which not only demonstrates the versatility of the DDH assumption in the area of 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 helps 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.

In addition, we presented a semi-generic conversion from a category mqPMT called Sigma-mqPMT to mqRPMT* (a weaker version of mqRPMT), making the first step towards investigating relations between mqPMT and mqRPMT. As an application of such conversion, we obtained a mqRPMT* protocol from FHE which is suitable for the unbalanced setting. We left the general connection between mqPMT and mqRPMT as an open problem.

When conducting performance comparison, we find that a number of PSO implementations suffer from one or more of the following deficiencies: (i) rely on multiple libraries, but configurations are not well documented; (ii) require sophisticated parameters tuning, but optimized parameters are not explicitly given; (iii) codes are not faithful to protocols described in paper, such as insecure random oracle instantiation, incorrect thread number counting etc. Sometimes, even making these programs successfully running would require tremendous efforts. We opensource C++ implementation with detailed documentations. We hope our implementation is useful for a high-quality MPC platform that admits easy and fair benchmarking of all PSO protocols.

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.

[Ber06] Daniel J. Bernstein. Curve25519: New diffie-hellman speed records. In *Public Key Cryptography - PKC 2006*, volume 3958 of *Lecture Notes in Computer Science*, pages 207–228. Springer, 2006.

[BHKL13] Daniel J. Bernstein, Mike Hamburg, Anna Krasnova, and Tanja Lange. Elligator: elliptic-curve points indistinguishable from uniform random strings. In *2013 ACM SIGSAC Conference on Computer and Communications Security, CCS 2013*, pages 967–980. ACM, 2013.

[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.

[Blo70] Burton H. Bloom. *Commun. ACM*, 13(7):422–426, 1970.

[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.

[CGT12] Emiliano De Cristofaro, Paolo Gasti, and Gene Tsudik. Fast and private computation of cardinality of set intersection and union. In *Cryptology and Network Security, 11th International Conference, CANS 2012*, volume 7712, pages 218–231. Springer, 2012.

[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.

[DRRT18] Daniel Demmler, Peter Rindal, Mike Rosulek, and Ni Trieu. PIR-PSI: scaling private contact discovery. *Proc. Priv. Enhancing Technol.*, 2018(4):159–178, 2018.

[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.

[FMV19] Daniele Friolo, Daniel Masny, and Daniele Venturi. A black-box construction of fully-simulatable, round-optimal oblivious transfer from strongly uniform key agreement. In *Theory of Cryptography - 17th International Conference, TCC 2019*, volume 11891 of *Lecture Notes in Computer Science*, pages 111–130. Springer, 2019.

[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.

[HEK12] Yan Huang, David Evans, and Jonathan Katz. Private set intersection: Are garbled circuits better than custom protocols? In *19th Annual Network and Distributed System Security Symposium, NDSS 2012*, 2012.

[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.

[HLS+16] Kyle Hogan, Noah Luther, Nabil Schear, Emily Shen, David Stott, Sophia Yakoubov, and Arkady Yerukhimovich. Secure multiparty computation for cooperative cyber risk assessment. In *IEEE Cybersecurity Development, SecDev 2016*, pages 75–76. IEEE Computer Society, 2016.

[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.

[KLS+17] Ágnes Kiss, Jian Liu, Thomas Schneider, N. Asokan, and Benny Pinkas. Private set intersection for unequal set sizes with mobile applications. *Proc. Priv. Enhancing Technol.*, (4):177–197, 2017.

[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.

[LV04] Arjen K. Lenstra and Tim Voss. Information security risk assessment, aggregation, and mitigation. In *Information Security and Privacy: 9th Australasian Conference, ACISP 2004*, volume 3108 of *Lecture Notes in Computer Science*, pages 391–401. Springer, 2004.

[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.

[NTL+11] Arvind Narayanan, Narendran Thiagarajan, Mugdha Lakhani, Michael Hamburg, and Dan Boneh. Location privacy via private proximity testing. In *Proceedings of the Network and Distributed System Security Symposium, NDSS 2011*. The Internet Society, 2011.

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

[Opeb] OpenMP. https://www.openmp.org/resources/openmp-compilers-tools/.

[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.

[PSTY19] Benny Pinkas, Thomas Schneider, Oleksandr Tkachenko, and Avishay Yanai. Efficient circuit-based PSI with linear communication. In *Advances in Cryptology - EUROCRYPT 2019*, volume 11478 of *Lecture Notes in Computer Science*, pages 122–153. 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.

[RA18] Amanda Cristina Davi Resende and Diego F. Aranha. Faster unbalanced private set intersection. In *Financial Cryptography and Data Security - 22nd International Conference, FC 2018*, volume 10957 of *Lecture Notes in Computer Science*, pages 203–221, 2018.

[Rab05] Michael O. Rabin. How to exchange secrets with oblivious transfer. 2005. 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.

[RR22] Srinivasan Raghuraman and Peter Rindal. Blazing fast PSI from improved OKVS and subfield VOLE. In *ACM CCS 2022*, 2022.

[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.

[RT21] Mike Rosulek and Ni Trieu. Compact and malicious private set intersection for small sets. In *CCS '21: 2021 ACM SIGSAC Conference on Computer and Communications Security*, pages 1166–1181. ACM, 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.

[TKC07] Juan Ramón Troncoso-Pastoriza, Stefan Katzenbeisser, and Mehmet Utku Celik. Privacy preserving error resilient dna searching through oblivious automata. In *Proceedings of the 2007 ACM Conference on Computer and Communications Security, CCS 2007*, pages 519–528. ACM, 2007.

[ZCL+23] Cong Zhang, Yu Chen, Weiran Liu, Min Zhang, and Dongdai Lin. Optimal private set union from multi-query reverse private membership test. In *USENIX 2023*, 2023. 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

By plugging in DDH-based OPRF to the above generic construction, we get an instantiation of Sigma-mqPMT based on the DDH assumption (as shown in Figure 14).

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

Figure 14: Sigma-mqPMT based on the DDH assumption

B.2 Sigma-mqPMT from FHE

We then present an instantiation of Sigma-mqPMT (shown in Figure 15) based on oblivious polynomial evaluation (OPE), which in turn efficiently built from FHE. The obtained Sigma-mqPMT is actually the backbone of the unbalanced PSI protocol [CLR17].

$$
\begin{array}{ccc}
P_1 \text{ (server)} & & P_2 \text{ (client)} \\
Y = (y_1, \dots, y_{n_1}) & & X = (x_1, \dots, x_{n_2}) \\
a \leftarrow \perp & \overset{\perp}{\dashrightarrow} & \\
 & \xleftarrow{\vec{q} = \{q_1, \dots, q_{n_2}\}} & q_i \leftarrow \mathsf{FHE.Enc}(pk, x_i) \\
r_i \xleftarrow{\text{R}} \mathbb{F} & & \\
f_i \leftarrow r_i \prod_{y \in Y} (y_i - x) & \xrightarrow{\vec{z} = \{z_1, \dots, z_{n_2}\}} & e_i := \mathsf{FHE.Dec}(dk, z_i) \overset{?}{=} 0 \\
z_i \leftarrow \mathsf{FHE.Eval}(pk, f_i, q_i) & &
\end{array}
$$

Figure 15: Sigma-mqPMT based on FHE

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 $\{y_{\pi(1)}^s, \dots, y_{\pi(n)}^s\}$.

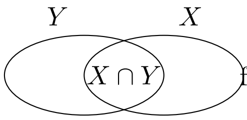
Clearly, Sim_{P_2}'s simulated view is identical to the real view.

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 $(\beta_1^s, \ldots, \beta_n^s)$, where $\mathsf{H}(x_i) = \beta_i$.



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

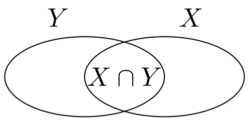
Clearly, Sim_{P_1}'s simulated view in Hybrid_1 is identical to P_1's real view.

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 $(g^{c_1}, \ldots, g^{c_n})$ 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 $g^a, g^{b_1}, \ldots, g^{b_n}, g^{c_1}, \ldots, g^{c_n})$ 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 $(g^{c_1}, \ldots, g^{c_n})$.



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. $\square$

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.