Private Set Operations from Multi-Query Reverse Private Membership Test

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

Abstract

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

We demonstrate the practicality of our framework with implementations. By plugging our cwPRF-based mqRPMT to the general framework, we obtain the first PSU protocol with strict linear complexity. For input sets of size 2^{20}, the resulting PSU protocol requires roughly 80 MB bandwidth, and 50 seconds using 8 threads. To the best of our knowledge, it requires the least communication among all the known PSU protocols. By plugging our FHE-based mqRPMT* to the general framework, we obtain a PSU* 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, zm_min, minglang_dong}@mail.sdu.edu.cn
†SKLOIS, IIE, Chinese Academy of Sciences. Email: zhangcong@iie.ac.cn

Contents

1 Introduction

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

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

Private computing on set intersection. Many real-world applications requires only partial/aggregate information about the intersection to be revealed. In this case we need private computing on set intersection (PCSI), including PSI-card for intersection cardinality [HFH99] and PSI-sum for intersection sum [IKN+20, MPR+20].

Private set union. Like PSI, PSU also has numerous applications in practice, such as cyber risk assessment and management via joint IP blacklists and joint vulnerability data. There are two paradigms of existing PSU protocols. The first is mainly based on public-key techniques [KS05, Fri07, HN10, DC17]. The second is mainly based on symmetric-key techniques [KRTW19, GMR+21].

PSO protocols are primarily designed for the balanced setting, where $|X| \approx |Y|$. In unbalanced setting, we have $|Y| \ll |X|$. PSI, PCSI and PSU are closely related functionalities. Among them, PSI is extensively studied. The state-og-the-art PSI is almost as efficient as the naive insecure hash-based protocol. In contrast to the affairs of PSI, the efficiency of the state-of-the-art PCSI and PSU are less satisfactory. In balanced setting, there are PSI protocols [Mea86, CM20] and PCSI protocols [IKN+20] with optimal linear complexity. In unbalanced setting, there are PSI protocols [CLR17, CHLR18, CMdG+21] with sublinear complexity in the larger set size $|X|$, but no such PCSI protocol is known. So far, there is no PSU protocol with linear complexity in either balanced or unbalanced setting in the literature. As to practical efficiency, PSI-card is concretely about 20× slower and requires over 30× more communication than PSI. and PSU is concretely about 20× slower and requires over 30× more communication than PSI.

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

1.1 Motivation

The above discussion indicates that the most efficient PSI protocols may not be easily adapted to PCSI and PSU protocols. Therefore, different approaches are employed for different private set operations, creating much more engineering effort. We are motivated to seek for a common core protocol that enables all private set operations, with the hope to design PSO in a unified framework. Moreover, given the huge efficiency gap between PSI and other closely related protocols, we are also motivated to give efficient construction of the core protocol to close the gap. In summary, it is intriguing to know:

Is there a core protocol that enables a unified framework for all private set operations? If so, can we give efficient constructions that lead to optimal complexity?

1.2 Our Contribution

In this work, we make positive progress on 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 private set operations. More precisely, mqRPMT itself already implies PSI-card; by coupling with OTe, mqRPMT implies PSI and PSU; by further coupling with secret sharing or additively homomorphic encryption, mqRPMT implies PSI-sum. Therefore, mqRPMT enables a PSO framework, which can perform all set operations in a unified and flexible manner.

Efficient construction of mqRPMT. We propose two generic constructions of mqRPMT. The first is based on a new cryptographic primitive called commutative weak PRF, the second is based on another new secure protocol called permuted oblivious PRF. Both of them can be instantiated from the DDH like assumptions in the random oracle model, leading to incredibly simple mqRPMT protocols with linear communication and computation complexity. Note that the asymptotic complexity of our PSO framework is dominated by the underlying mqRMPT. Therefore, all PSO protocols derived from our framework inherit linear complexity. Particularly, to the best of our knowledge, it is the first time to have PSU with linear complexity.

Relaxed mqRPMT. We propose a slight weak version of mqRPMT (denoted mqRPMT* hereafter). Compared to the standard mqRPMT, mqRPMT* allows the sender learn the intersection size. We show that mqRMPT* can be build from a special category of mqPMT called Sigma-mqPMT in a black-box manner via the "permute-then-test" recipe. This makes the initial step towards to establishing the connection between mqRPMT and mqPMT. By instantiating the transformation from fully homomorphic encryption (FHE) based Sigma-mqPMT, we obtain an efficient mqRPMT* in unbalanced setting, which immediately gives rise to a PSU* protocol with sublinear complexity of the size of larger set X.

Evaluations. We implement our framework. The experimental results demonstrate that our PSU protocol is superior to all the known PSU protocols in terms of communication cost.

1.3 Technical Overview

PSO from mqRPMT. As discussed above, mqPMT (a.k.a. PSI) protocols generally is not applicable for computing PCSI and PSU. We examine the reverse direction, i.e., whether the core protocol underlying PSU can be used for computing PSI and PCSI. We identify that the central protocol beneath all the existing PSU protocols is mqRPMT, which is a generalization of RPMT proposed in [KRTW19]. Roughly speaking, mqRPMT is a two party protocol between a client with set $X = (x_1, \ldots, x_n)$ and a server with set Y. After execution of the protocol, the server learns an indication bit vector $(e_1, \ldots, e_n)$ such that $e_i = 1$ if and only if $x_i \in Y$ but without knowing x_i, while the client learns nothing. Superficially, mqRPMT is similar to mqPMT, except that the server but not the client learns the test results instead. This subtle difference turns out to be significant. To see this, note that in mqRPMT the information of intersection (except its cardinality) is hidden from both sides, while in mqPMT the intersection is finally known by the client. In light of this difference, mqRPMT is particular suitable for functionalities that have to keep intersection private. A PSU protocol is immediate by having the sender (play the role of client) and the receiver (play the role of server) invoke a mqRPMT protocol on the first place, then carrying out n one-out-two OT with e_i and $(\perp, y_i)$ respectively. PSI and other protocols such as PSI-card and PSI-sum can be constructed similarly by coupling with additively homomorphic encryption or secret sharing.

The seminal PSI protocol [Mea86] (related ideas were appeared in [Sha80, HFH99]) is based on the commutative properties of the DH function, known as DH-PSI. After roughly four decades, the DH-PSI protocol is still the most easily understood and to implement one among numerous PSI protocols. It is somewhat surprisingly that no counterpart is known in the PSU setting yet. An intriguing question is: Can DH strike back? In this work, we give an affirmative answer to the above question.

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

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

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

We show that cwPRF can be realized from DDH-like assumptions. Henceforth, DDH strikes back with a simple PSU protocol, and demonstrates the DH-function endowed with versatile properties is truly a golden goose in cryptography.

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

The question remains is how to build pOPRF. One common approach to build OPRF is "mask-then-unmask". We choose OPRF as the starting point. The rough idea is exploiting the input homomorphism to mask inputs[1], then unmask the outputs. If the mask procedure is different per input, then the unmask procedure must be carried out accordingly. Therefore, OPRF protocols of this case cannot be easily adapted to pOPRF, cause the receiver is unable to perform the unmask procedure over permuted masked outputs correctly, namely, recovering outputs in permuted order. The above analysis indicates us that if the masking procedure can be done via a unifying manner, then the receiver might be able

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

to unmask the permuted masked outputs correctly. Observe that the simplest way to perform unified masking is to apply a weak pseudorandom function G_s to the intermediate input $\mathsf{H}(x)$. To enable efficient unmask procedure, we further require that G_s is a permutation and commutative with respect to F_k. This yields a simple pOPRF construction from enhanced cwPRF (F_k, G_s) in which $G_s(\cdot)$ is a weak pseudorandom permutation. More precisely, to build pOPRF, the sender picks a random PRF key k for F, while the receiver with input $X = (x_1, \ldots, x_n)$ picks a random PRP key s for G. The receiver then sends $\{G_s(\mathsf{H}(x_i))\}_{i\in[n]}$ to the sender. Upon receiving the masked intermediate inputs, the sender applies F_k to them, then sends the results in permuted order, a.k.a. $\{F_k(G_s(\mathsf{H}(x_{\pi(i)}))\}_{i\in[n]}$. Finally, the receiver applies G_s^{-1} to the permuted masked outputs, and will obtain $\{F_k(\mathsf{H}(x_{\pi(i)})\}_{i\in[n]}$ by the commutative property.

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

mqRPMT* from mqPMT. Towards the goal of studying the connection between mqRPMT and mqPMT, we abstract a category of mqPMT protocol called Sigma-mqPMT, which underlies several PSI protocols [Mea86, CLR17] with linear complexity. Following the permute-then-test approach, we can tweak Sigma-mqPMT to mqRPMT* with same asymptotic complexity, and thus obtain asymptotically efficient PSU* protocols in both balanced and unbalanced settings. We leave the more general connection as an interesting open problem.

Applications of mqRPMT. With mqRPMT in hand, we can build a general PSO framework. mqRPMT itself immediately give rise to private set intersection/union cardinality. Coupling with oblivious transfer, we can obtain PSI, private set intersection sum or PSU, depending the messages on OT sender's side.

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

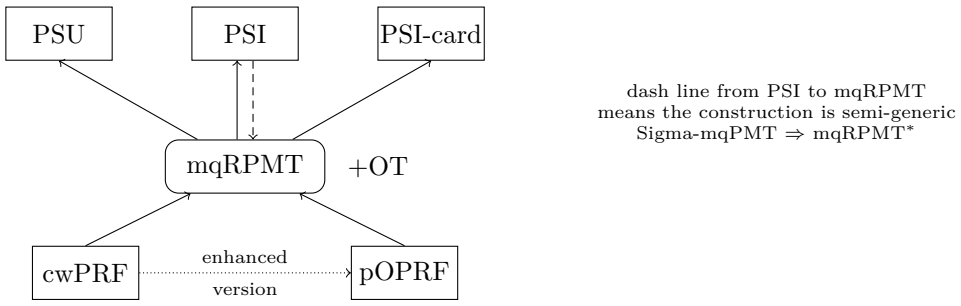


Figure 1: Constructions and Applications of mqRPMT

1.4 Related Works

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 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 protocols and the resulting PSO protocols. In a concurrent work [ZCL+22], Zhang et al. [ZCL+22] build mqRPMT protocols with linear complexity from oblivious key-value store, set-membership encryption and oblivious vector decryption-then-test functionality. Compared to their construction, our protocol is conceptually simpler. Besides, we explore more applications of mqRPMT, while they only focus on PSU protocol.

2 Preliminaries

Notations. We assume that every set X has a default order (e.g. lexicographical order), and write it as $X = \{x_1, \ldots, x_n\}$. We use $(x_1, \ldots, x_n)$ to deonote a vector, whose ith element is x_i.

2.1 MPC in the Semi-honest Model

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

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

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

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

2.2 Private Set Operation

PSO is a special case of secure two-party computation.

Parameters: size of sets n.

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

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

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

3 Protocol Building Blocks

3.1 Oblivious Transfer

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

3.2 Multi-Query RPMT

RPMT [KRTW19] refers to a protocol where the client with input x interacts with a server holding a set Y. As a result, the server learns (only) the bit indicating whether $x \in Y$, while the client learns nothing about the set Y. The default notion of RPMT allows the client to query for a single element. While this procedure can be repeated several times, one may seek more efficient solutions allowing the client to

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

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

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

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

make n distinct queries at a reduced cost. This generalized notion of n-time RPMT is straightforward to define. Hereafter, we refer to n-time RPMT as multi-query RMPT. In Figure 4 we formally define the ideal functionality for mqRPMT. We also define a relaxed version of mqRPMT called mqRPMT*, in which the client is given $|X \cap Y|$.

Parameters: number of RPMT queries n.

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

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

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

4 Review of Pseudorandom Function

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

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

- $\mathsf{Setup}(1^\kappa)$: on input a security parameter κ, outputs public parameter pp. pp specifies a family of keyed functions $F : K \times D \to R$, 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 PRFs is pseudorandomness.

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

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

where $\mathcal{O}_{\mathsf{ror}}(0, x) = F_k(x)$, $\mathcal{O}_{\mathsf{ror}}(1, x) = \mathsf{H}(x)$ (here H is chosen uniformly at random from all the functions from D to R[2]). Note that $\mathcal{A}$ can adaptively access the oracle $\mathcal{O}_{\mathsf{ror}}(\beta, \cdot)$ polynomial many times. We say

[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.

that F is pseudorandom if for any PPT adversary its advantage function $\mathsf{Adv}_{\mathcal{A}}(\kappa)$ is negligible in κ. We refer to such security as full PRF security.

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

4.1 Weak PRF from the DDH Assumption

We build weak PRF from the DDH assumption 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 x^k.
- $\mathsf{KeyGen}(pp)$: outputs $k \xleftarrow{\mathrm{R}} \mathbb{Z}_p$.
- $\mathsf{Eval}(k, x)$: 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. *The above construction is weak pseudorandom assuming the hardness of the DDH problem.*

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

Setup: $\mathcal{B}$ sends $pp = (\mathbb{G}, g, p)$ to $\mathcal{A}$. $\mathcal{B}$ implicitly set 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 $F_k(\cdot)$. □

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

4.2 PRF from the DDH Assumption

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

For completeness, we provide the details as below.

- $\mathsf{Setup}(1^\kappa)$: runs $\mathsf{GroupGen}(1^\kappa) \to (\mathbb{G}, g, p)$, pick 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 $k \xleftarrow{R} \mathbb{Z}_p$.
- $\mathsf{Eval}(k, x)$: on input $k \in \mathbb{Z}_p$ and $x \in D$, outputs $\mathsf{H}(x)^k$.

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

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

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

- Setup: $\mathcal{B}$ sends $pp = (\mathbb{G}, g, p, \mathsf{H})$ to $\mathcal{A}$, and implicitly sets a as the key of PRF.
- Random oracle query: for random oracle query $\langle x_i \rangle$, $\mathcal{B}$ programs $\mathsf{H}(x_i) := g^{b_i}$.
- Real-or-random query: without loss of generality, it is safe to assume adversary has already make 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 build from weak pseudorandom group action (c.f. Definition in Appendix A.1) in a similar way.

5 Commutative Weak Pseudorandom Functions

5.1 Definition of Commutative Weak PRF

We first formally define two standard properties for keyed functions.

Composable. For a family of keyed function F, 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 function, we say it is commutative if:

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

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

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

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

$$\forall k \in K, r \in R, \forall x \in X : F_k(G_s(x)) = G_s(F_k(x))$$

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

Remark 5.1. We note that our notion if commutative weak PRF (cwPRF) is similar to but strictly weaker than a previous notion called commutative encryption [AES03]. The difference is that cwPRF neither require 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 $\mathsf{H} : \{0,1\}^\ell \to D$.

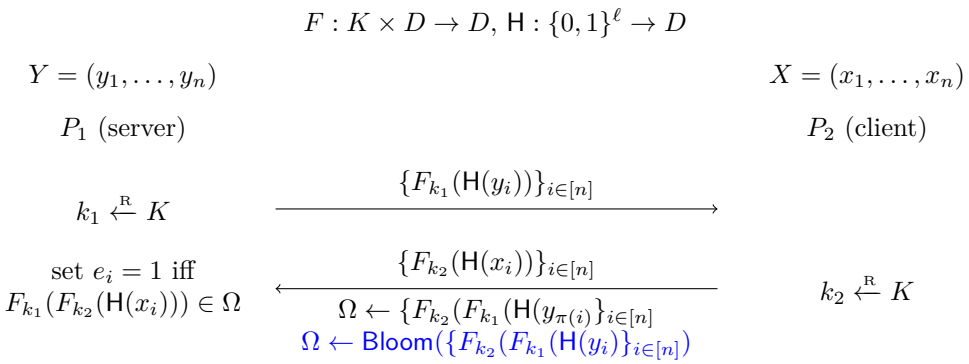
Parameters:

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

Protocol:

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

Figure 5: Multi-query RPMT from commutative weak PRF



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

of F, we have $\Pr[E_{10}] = \Pr[E_{11}] = 2^{-\ell}$. Therefore, we have $\Pr[E] \leq \Pr[E_0] + \Pr[E_{10}] + \Pr[E_{11}] = 2^{-\sigma} + 2^{-\ell+1}$.

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

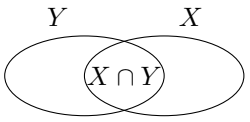
Proof. We exhibit simulators Sim_{P_1} and Sim_{P_2} for simulating corrupt P_1 and P_2 respectively, and argue the indistinguishability of the simulated transcript from the real execution. Let $|X \cap Y| = m$.

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.

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



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

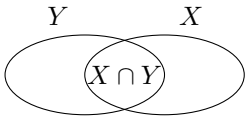
Clearly, Sim_{P_2}'s simulated view in Hybrid_1 is identical to P_2's real view.

Hybrid_2: Sim_{P_2} does not choose the randomness for P_1 (i.e., picks $k_1 \xleftarrow{R} K$), and simulates without the knowledge of Y. It emulates the random oracle H honestly as before, and only changes the simulation of P_1's message.

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

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



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

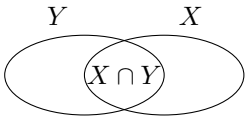
If $\eta_i = F_k(\beta_i)$ for $i \in [n]$, then $\mathcal{A}$'s simulation is identical to Hybrid_1. If η_i are random values, then $\mathcal{A}$'s simulation is identical to Hybrid_2.

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

Hybrid_0: P_1's view in the real protocol.

Hybrid_1: Given P_1's input Y and output $(e_1, \ldots, e_n)$, Sim_{P_1} chooses the randomness for P_2 (i.e., picks $k_2 \xleftarrow{R} K$ and a random permutation π over $[n]$), and simulates with the knowledge of X.

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

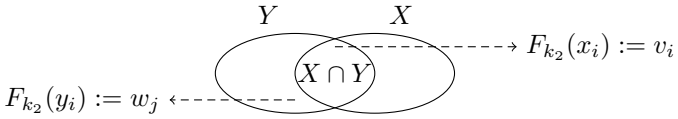


for $z_i \in D$, $\mathsf{H}(z_i) := \alpha_i \xleftarrow{R} D$

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

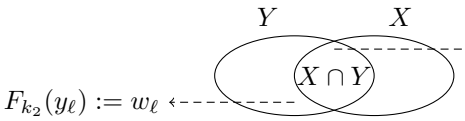
Hybrid_2: Sim_{P_1} does not choose randomness for P_1, and simulates without the knowledge of X. It simulates the random oracle H honestly as before, and changes it simulation of P_2's message. Let m be the Hamming weight of $(e_1, \ldots, e_n)$.

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



We argue that the view in Hybrid_1 and Hybrid_2 are computationally indistinguishable. More precisely, a PPT adversary $\mathcal{A}$ (with knowledge of X and Y) against cwPRF are given $2n-m$ tuples (γ_i, η_i) where $\gamma_i \xleftarrow{R} D$, 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 $k_1 \xleftarrow{R} K$.

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



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

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. □

6 mqRPMT from Permuted Oblivious Pseudorandom Function

6.1 Definition of Permuted OPRF

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

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

Parameters: number of OPRF queries n.

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

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

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

6.2 Construction of Permuted OPRF

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

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

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

Figure 7: Permuted OPRF from cwPRF

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

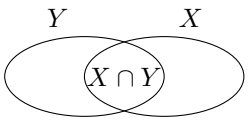
Proof. We exhibit simulators Sim_{P_1} and Sim_{P_2} for simulating corrupt P_1 and P_2 respectively, and argue the indistinguishability of produced transcript from the real execution.

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

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

Hybrid$_1$: Given P_1's output k and π, Sim_{P_1} chooses the randomness r 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 $\alpha_i \xleftarrow{R} \mathbb{G}$ and assigns $\mathsf{H}(z_i) := \alpha_i$.
- Sim_{P_1} outputs $(\beta_1^r, \ldots, \beta_n^r)$, where $\mathsf{H}(x_i) = \beta_i$.



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

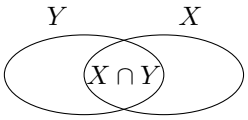
Clearly, Sim_{P_1}'s simulated view in Hybrid_1 is identical to P_1's real view.

Hybrid_2: Sim_{P_1} does not choose the randomness for P_2, and simulates without the knowledge of X. It honestly emulates random oracle H as in Hybrid_1, and only changes the simulation of P_2's message.

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

We argue that the view in Hybrid_1 and Hybrid_2 are computationally indistinguishable. Let $\mathcal{A}$ be a PPT adversary against the weak pseudorandom of G_s. Given a real-or-random oracle $\mathcal{O}_{\mathsf{ror}}(\cdot)$, $\mathcal{A}$ is asked to distinguish which mode he is in. $\mathcal{A}$ queries the $\mathcal{O}_{\mathsf{ror}}(\cdot)$ n times, and obtains (γ_i, η_i) in return. $\mathcal{A}$ then simulates (with the knowledge of X) as below:

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



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

Clearly, if $\eta_i = G_s(\gamma_i)$, $\mathcal{A}$ simulates Hybrid_1. Else, it simulates Hybrid_2. Thereby, Sim_{P_1}'s simulated view is computationally indistinguishable to P_1's real view.

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

Hybrid_0: P_2's view in the real protocol.

Hybrid_1: Given P_2's input $X = (x_1, \ldots, x_n)$ and output $\{F_k(\mathsf{H}(x_{\pi(i)}))\}_{i\in[n]}$, Sim_{P_2} emulates the random oracle H honestly, picks $r \xleftarrow{\text{R}} \mathbb{Z}_p$, simulates message from P_1 as $\{G_s(F_k(\mathsf{H}(x_{\pi(i)})))\}_{i\in[n]}$.

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

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

Parameters:

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

Protocol:

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

Figure 8: Permuted OPRF from the DDH assumption

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

6.3 mqRPMT from Permuted OPRF

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

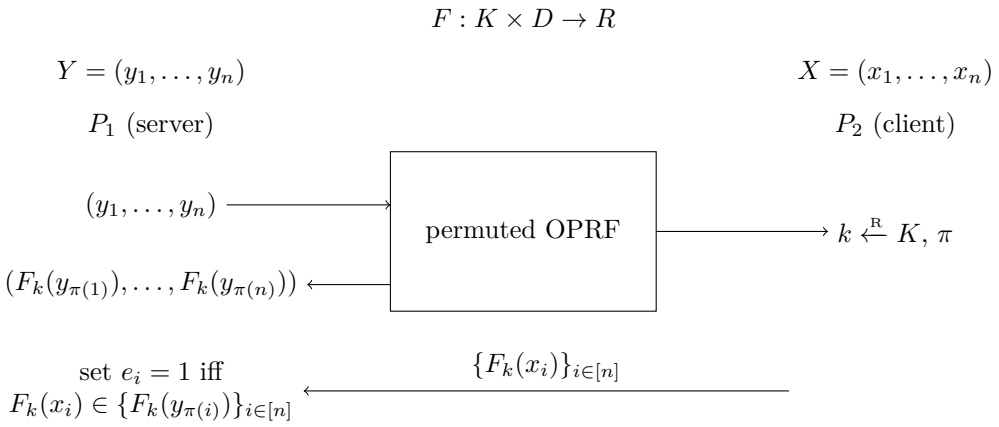
Parameters:

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

Protocol:

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

Figure 9: mqRPMT from permuted OPRF



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

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

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

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

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

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

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

Hybrid_0: P_1's view in the real protocol. Note that P_1's view consists of its view in stage 1 (the permuted OPRF part) and its view in stage 2.

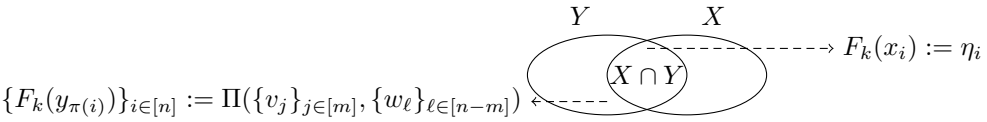
Hybrid_1: Given P_1's input $Y = (y_1, \ldots, y_n)$ and output $(e_1, \ldots, e_n)$, Sim_{P_1} creates the simulated view as below:

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

The simulated stage 2'view is identical to that in the real one. By the semi-honest security of permuted OPRF on P_1' side, the stage 1's simulated view of stage 1 in is computationally indistinguishable to that in the real one. Thereby, the simulated view in Hybrid_1 is computationally indistinguishable to the real one.

Hybrid_2: Sim_{P_1} creates the simulated view without the knowledge of X, and it neither picks k nor explicitly picks π:

- generate stage 2's view by outputting $(\eta_1, \ldots, \eta_n)$, 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 $\{w_\ell\}_{\ell\in[n-m]}$; pick a random permutation Π of $(\{v_j\}_{j\in[m]}, \{w_\ell\}_{\ell\in[n-m]})$, treat the result as $(F_k(y_{\pi(1)}), \ldots, F_k(y_{\pi(n)}))$ (note that the real permutation π is unknown to the simulator cause it does not know $X \cap Y$); then generate its stage 1's view by invoking the simulator for P_1 of permuted OPRF with input $(y_1, \ldots, y_n)$ and output $(F_k(y_{\pi(1)}), \ldots, F_k(y_{\pi(n)}))$.



We argue that the simulated views in Hybrid_1 and Hybrid_2 are computationally indistinguishable based on the pseudorandomness of F. Let $\mathcal{A}$ be an adversary against F. Given X and Y, $\mathcal{A}$ simulates as below:

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

Clearly, if $\mathcal{A}$ queries the real oracle, then its simulation is identical to that Hybrid_1. Else, its simulation is identical to that Hybrid_2. This reduces the computational indistinguishability of views in Hybrid_1 and Hybrid_2 to the pseudorandomness of $F_k(\cdot)$. Therefore, Sim_{P_1}'s simulation is indistinguishable to the real one.

This proves the theorem. □

7 mqRPMT from Sigma-mqPMT

7.1 Private Membership Test

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

Sigma-PMT proceeds via following pattern.

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

After receiving t, client P_2 can decide if $x \in Y$ by running $\mathsf{Test}(a, x, q, t)$. 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 seed 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-free:** Each test query q_i is only related to the element x_i under test and the randomness of P_2.

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

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

We consider mqRPMT built from Sigma-PMT that is reusable, non-adaptive and supports stateless testing, and refer it to Sigma-mqPMT, as depicted in Figure 10.

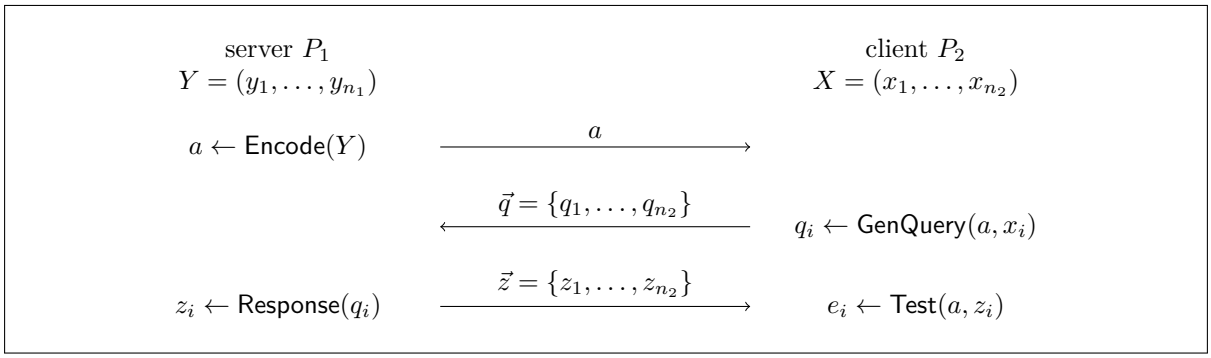


Figure 10: Sigma-mqPMT

We assume the simulator $\mathsf{Sim}(X, \vec{e})$ for client P_2 is composed of two sub-routines $(\mathsf{Sim}', \mathsf{Sim}'')$, and satisfies the following properties:

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

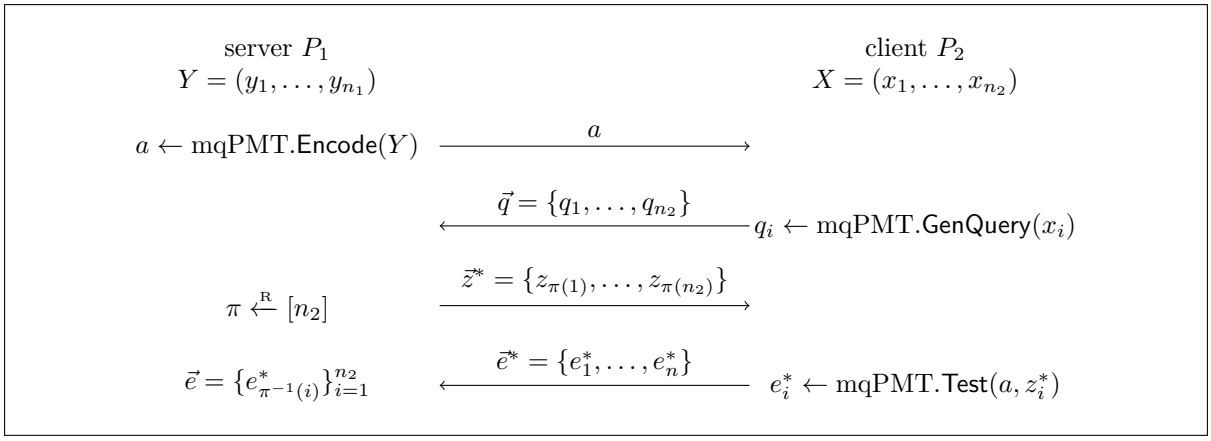


Figure 11: mqRPMT* from Sigma-mqPMT

7.2 Connection to Sigma-mqPMT

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

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

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

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

Hybrid_0: P_2's view in the real protocol.

Hybrid_1: Sim_{P_2} chooses the randomness for P_1, and simulates with the knowledge of Y. Clearly, Sim_{P_2}'s simulation is identical to the real view of P_2.

Hybrid_2: Sim_{P_2} does not choose the randomness for P_1, and simulates without the knowledge of Y. Instead, it invokes the Sigma-mqPMT's simulator for P_2 on his private input X and output $\vec{e}^*$ to emulate the view $(a, \vec{z}^*)$ in the following manner:

- for $1 \leq i \leq n_2$, run $\mathsf{Sim}'(e^*_i; r_i) \to z^*_i$, obtaining $\vec{z}^* = (z^*_1, \ldots, z^*_n)$.
- run $\mathsf{Sim}''(\{(e^*_i, r_i\}_{i \in [n_2]}; s) \to a$.

By the *locality* and *order invariance* properties, the simulated view in Hybrid_2 and Hybrid_1 are computationally indistinguishable based on semi-honest security of mqPMT on P_2 side.

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

Hybrid_0: P_1's view in the real protocol.

Hybrid_1: Sim_{P_1} chooses the randomness for P_2, and simulates with the knowledge of X. Clearly, Sim_{P_1}'s simulation is identical to the real view of P_1.

Hybrid_2: Sim_{P_1} does not choose the randomness for P_2, and simulates without the knowledge of X. Instead, it first invokes the Sigma-mqPMT's simulator for P_1 on input Y to generate $\vec{q}$, then picks a random permutation π, 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. □

8 Applications of mqRPMT

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

Parameters:

- Input of receiver P_1: $Y = \{y_1, \ldots, y_{n_1}\} \subseteq \{0,1\}^\ell$.
- Input of sender P_2: $X = \{x_1, \ldots, x_{n_2}\} \subseteq \{0,1\}^\ell$ and $V = \{v_1, \ldots, v_{n_2}\}$.

Protocol:

0. P_2 shuffles the set $(x_1, \ldots, x_{n_2})$ and $(v_1, \ldots, v_{n_2})$ according to the same random permutation over $[n_2]$. For simplicity, we still use the original notation to denote the vector after permutation.
1. P_1 (playing the role of server) with Y and P_2 (playing the role of client) with $X = \{x_1, \ldots, x_{n_2}\}$ invoke $\mathcal{F}_{\mathsf{mqRPMT}}$. P_1 obtains an indication bit vector $\vec{e} = (e_1, \ldots, e_{n_2})$. P_2 obtains nothing.
   - **cardinality:** P_1 learns the cardinality by calculating the Hamming weight of $\vec{e}$.
2. P_1 and P_2 invoke n_2 instances of OT via $\mathcal{F}_{\mathsf{OT}}$. P_1 uses $\vec{e}$ as the choice bits.
   - **intersection:** P_2 uses $(\perp, x_i)$ as input to the ith OT. P_1 learns $\{x_i \mid e_i = 1\}_{i\in[n_2]} = X \cap Y$.
   - **union:** P_2 uses $(x_i, \perp)$ as input to the ith OT. P_1 learns $\{x_i \mid e_i = 0\}_{i\in[n_2]} = X/Y$, and outputs $\{X/Y\} \cup Y = X \cup Y$.
   - **intersection sum with cardinality:** P_2 randomly generate r_i subject to the constraint $\sum_{i=1}^{n_2} r_i = 0$, then uses $(r_i, r_i + v_i)$ as input to the ith OT. P_1 learns $\{\sum_{i=1}^{n_2} v_i \mid e_i = 1\}_{i\in[n_2]} = X \cap Y$.
   - **intersection sum with cardinality*:** P_2 generates a key pair (pk, dk) for some AHE, then encrypts v_i to obtain $c_i \leftarrow \mathsf{Enc}(pk, v_i)$, uses $(\perp, c_i)$ as input to the ith OT. P_1 learns $\{\sum_{i=1}^{n_2} c_i \mid e_i = 1\}_{i\in[n_2]}$, then re-randomizes the sum ciphertext and sends it back to P_2. P_2 decrypts the ciphertext, obtaining the intersection sum.

Figure 12: PSO from mqRPMT

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

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

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

Hybrid$_0$: P_2's view in the real protocol. Note that P_2's view consists of two parts, i.e., the mqRPMT part of view (stage 1) and the OT part of view (stage 2).

Hybrid$_1$: Sim_{P_2} first invokes the simulator for client in the mqRPMT with X as input to generate the stage 1's part of view, then invokes the simulator for sender in the OT with $\{(x_i, \perp)\}_{i\in[n_2]}$ as input to generate stage 2's part of view. By the semi-honest security of mqRPMT on client side and the semi-honest security for OT on sender side, the simulation is indistinguishable to the real view via standard hybrid argument.

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

Hybrid$_0$: P_1's view in the real protocol. Note that P_1's view also consists of two parts, i.e., the mqRPMT part of view (stage 1) of and the OT part of view (stage 2).

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

- pick a random indication vector $\vec{e} = (e_1, \ldots, e_{n_2})$ with Hamming weight $m = X \cap Y$, then generate the output vector $\vec{z} = (z_1, \ldots, z_{n_2})$ 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 $e_i = 0$; then invoke the simulator for OT receiver with input $\vec{e}$ and output $\vec{z}$ and to generate stage 2's view.
- invoke the simulator for mqRPMT server with input Y and output $\vec{e} = (e_1, \ldots, e_{n_2})$ 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 $(x_1, \ldots, x_{n_2})$) in the real protocol. By the semi-honest security of mqRPMT on server side and the semi-honest security for OT on receiver side, the simulation is indistinguishable to the real view via standard hybrid argument.

This proves the theorem. □

9 Implementation

We will report the experimental results and comparison to related works soon.

Acknowledgments

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

References

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

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

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

[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.

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

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

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

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

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

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

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

[IKN+20] Mihaela Ion, Ben Kreuter, Ahmet Erhan Nergiz, Sarvar Patel, Shobhit Saxena, Karn Seth, Mariana Raykova, David Shanahan, and Moti Yung. On deploying secure computing: Private intersection-sum-with-cardinality. In *IEEE European Symposium on Security and Privacy, EuroS&P 2020*, pages 370–389. IEEE, 2020.

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

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

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

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

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

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

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

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

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

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

[PSZ14] Benny Pinkas, Thomas Schneider, and Michael Zohner. Faster private set intersection based on OT

extension. In *Proceedings of the 23rd USENIX Security Symposium, 2014*, pages 797–812. USENIX Association, 2014.

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

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

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

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

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

A Missing Definitions

A.1 Weak Pseudorandom EGA

We begin by recalling the definition of a group action.

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

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

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

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

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

1. The group $\mathbb{G}$ is finite and there exist PPT algorithms for:
   (a) Membership testing, i.e., to decide if a given bit string represents a valid group element in $\mathbb{G}$.
   (b) Equality testing, i.e., to decide if two bit strings represents the same group element in $\mathbb{G}$.
   (c) Sampling, i.e., to sample an element g from a uniform (or statistically close to) distritution 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 arbitrary 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 computable 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 $g \xleftarrow{R} \mathbb{G}$ and each $x_i, u_i \xleftarrow{R} X$.

A.2 mqPMT from OPRF

In Figure 13, we show how to build mqPMT from permuted OPRF $F : K \times D \to R$.

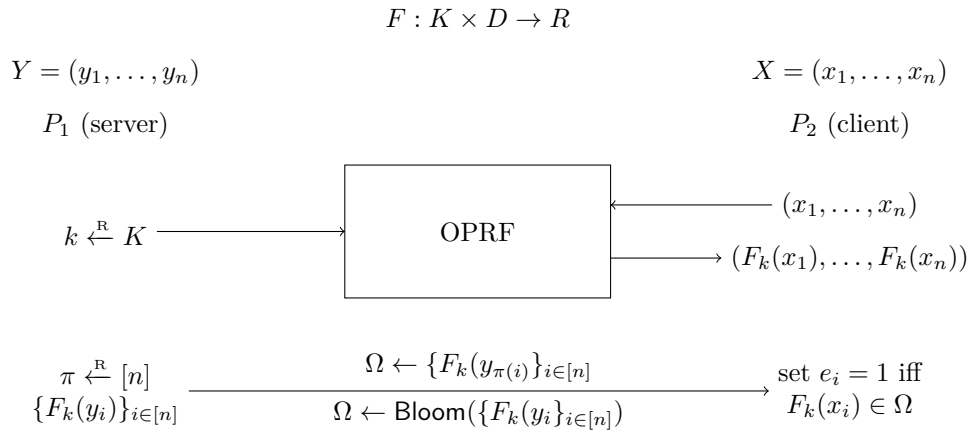
Parameters:

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

Protocol:

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

Figure 13: mqPMT from OPRF



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

Theorem A.1. *The above mqPMT protocol described in Figure 13 is secure in the semi-honest model assuming the security of OPRF F.*

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

Security against corrupt 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} simply picks k, then invokes the simulator for P_1 in the OPRF with k 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 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. Note that P_2's view consists of its view in stage 1 (the OPRF part) and its view in stage 2.

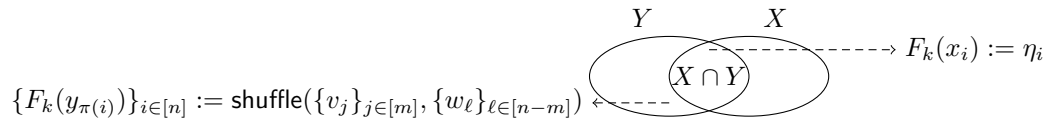
Hybrid_1: Given P_2's input $X = (x_1, \ldots, x_n)$ and output $(e_1, \ldots, e_n)$, Sim_{P_2} creates the simulated view as below:

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

The simulated stage 2's view is identical to that in the real one. By the semi-honest security of OPRF on P_2 side, the stage 1's simulated view is computationally indistinguishable to that in the real one. Thereby, the simulated view in Hybrid_1 is computationally indistinguishable to the real one.

Hybrid_2: Sim_{P_2} creates the simulated view without the knowledge of Y, and it neither picks k nor explicitly picks π:

- simulate stage 1's output by outputting $(\eta_1, \ldots, \eta_n)$, 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 $\{w_\ell\}_{\ell\in[n-m]}$; pick a random permutation of $(\{v_j\}_{j\in[m]}, \{w_\ell\}_{\ell\in[n-m]})$, treat the result as $(F_k(y_{\pi(1)}), \ldots, F_k(y_{\pi(n)}))$ (note that the real permutation π is unknown to the simulator cause it does not know the order of y_i); then generate its stage 1's view by invoking the simulator for P_2 of OPRF with input $(x_1, \ldots, x_n)$ and output $(\eta_1, \ldots, \eta_n)$.



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 with $(x_1, \ldots, x_n)$ and obtain $(\eta_1, \ldots, \eta_n)$, then generate stage 1's view by invoking the simulator for P_2 of OPRF with input $(x_1, \ldots, x_n)$ and outputs $(\eta_1, \ldots, \eta_n)$.
- pick a random permutation π, query the real-or-random oracle with $(y_1, \ldots, y_n)$ and obtains $(\zeta_1, \ldots, \zeta_n)$ in return, then generate stage 2's view by outputting $(\zeta_{\pi(1)}, \ldots, \zeta_{\pi(n)})$.

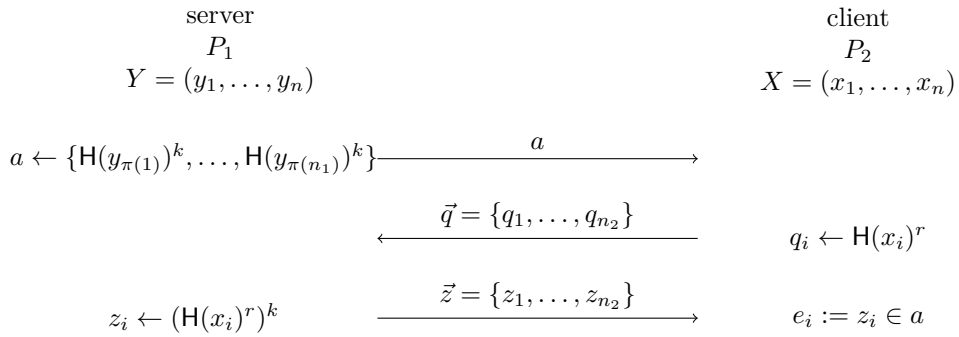
Clearly, if $\mathcal{A}$ queries the real oracle, then its simulation is identical to that Hybrid_1. Else, its simulation is identical to that Hybrid_2. This reduces the computational indistinguishability of views in Hybrid_1 and Hybrid_2 to the pseudorandomness of $F_k(\cdot)$. Therefore, Sim_{P_2}'s simulation is indistinguishable to the real one.

This proves the theorem. □

B Instantiations of Sigma-mqPMT

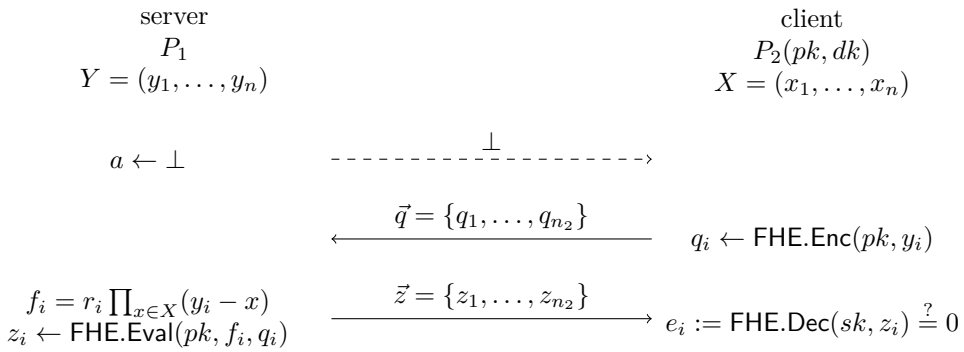
B.1 Sigma-mqPMT from DDH

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



B.2 Sigma-mqPMT from FHE

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



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: $\{\mathsf{AHE.Enc}(pk, y_i^1), \ldots, \mathsf{AHE.Enc}(pk, y_i^{n_1})\}$.

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

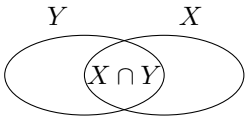
Clearly, Sim_{P_2}'s simulated view is identical to the real view.

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

Hybrid_0: P_1's view in the real protocol.

Hybrid_1: Given P_1's output k and π, Sim_{P_1} chooses the randomness s for P_2, and simulates with the knowledge of $X = (x_1, \ldots, x_n)$:

- RO queries: Sim_{P_1} honestly emulates random oracle H. For every query $\langle z_i \rangle$, picks $\alpha_i \xleftarrow{R} \mathbb{G}$ 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 $z_i \in D$, $\mathsf{H}(z_i) := \alpha_i \xleftarrow{R} D$

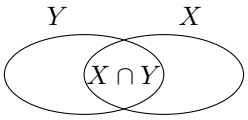
Clearly, Sim_{P_1}'s simulated view in Hybrid_1 is identical to P_1's real view.

Hybrid_2: Sim_{P_1} does not choose the randomness for P_2, and simulates without the knowledge of X. It honestly emulates random oracle H as in Hybrid_1, and only changes the simulation of P_2's message.

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

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 $a, b_i \xleftarrow{R} \mathbb{Z}_p$, $\mathcal{A}$ is asked to distinguish if $c_i = ab_i$ or random values. $\mathcal{A}$ implictly 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 $\alpha_i \xleftarrow{R} \mathbb{G}$ 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 $z_i \notin X$, $\mathsf{H}(z_i) := \alpha_i \xleftarrow{R} D$
for $z_i \in X$, $\mathsf{H}(z_i) := g^{b_i}$

Clearly, if $c_i = ab_i$, $\mathcal{A}$ simulates Hybrid_1. Else, it simulates Hybrid_2. Thereby, Sim_{P_1}'s simulated view is computationally indistinguishable to P_1's real view.

This proves the theorem. □

Remark C.1. In the above security proof, when establishing the security against corrupt sender, we can obtain a more modular proof by reducing the indistinguishability of simulated views in Hybrid_1 and Hybrid_2 to the pseudorandomness of $F_k(\mathsf{H}(\cdot))$, which is in turn based on the DDH assumption.