

# Private Set Operations from Multi-Query Reverse Private Membership Test

Yu Chen <sup>\*</sup>    Min Zhang<sup>\*</sup>    Cong Zhang <sup>†</sup>    Minglang Dong<sup>\*</sup>    Weiran Liu <sup>‡</sup>

## Abstract

Private set operations allow two parties to 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. The technical heart of our framework is multi-query reverse private membership test (mqRPMT) protocol [ZCL<sup>+</sup>23]. In mqRPMT, a client with a vector  $X = (x_1, \dots, x_n)$  interacts with a server holding a set  $Y$ . As a result, the server only learns a bit vector  $(e_1, \dots, e_n)$  indicating whether  $x_i \in Y$  but without knowing the value of  $x_i$ , while the client learns nothing. We present two constructions of mqRPMT from newly introduced cryptographic primitive and protocol. One is based on commutative weak pseudorandom function (cwPRF), the other is based on permuted oblivious pseudorandom 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 weak version of mqRPMT dubbed mqRPMT<sup>\*</sup>, in which the client also learns the cardinality of  $X \cap Y$ . We show that mqRPMT<sup>\*</sup> 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 to the general 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 in running time and a 10.9 – 14.8 $\times$  shrinking in communication cost. For cardinality-with-sum functionality, our protocol achieves a 28.5 – 76.3 $\times$  speedup in running time and 7.4 $\times$  shrinking in communication cost. For union functionality, our protocol is the first one that attains strict linear complexity. It requires the least concrete computation and communication costs in all settings, achieving a 2.7 – 17 $\times$  speedup in running time and 2 $\times$  shrinking in communication cost. Concretely, for input set 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. For private-ID functionality, our protocol achieves a 2.7 – 4.9 $\times$  speedup in running time. Moreover, by plugging our FHE-based mqRPMT<sup>\*</sup> to the general framework, we obtain a PSU<sup>\*</sup> 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

---

<sup>\*</sup>Shandong University. Email: [yuchen.prc@gmail.com](mailto:yuchen.prc@gmail.com), [{zm\\_min, minglang\\_dong}@mail.sdu.edu.cn](mailto:{zm_min, minglang_dong}@mail.sdu.edu.cn)

<sup>†</sup>SKLOIS, IIE, Chinese Academy of Sciences. Email: [zhangcong@iie.ac.cn](mailto:zhangcong@iie.ac.cn)

<sup>‡</sup>Alibaba Group. Email: [weiran.lwr@alibaba-inc.com](mailto:weiran.lwr@alibaba-inc.com)

# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	Motivation . . . . .	2
1.2	Our Contribution . . . . .	2
1.3	Technical Overview . . . . .	2
1.4	Related Works . . . . .	5
<b>2</b>	<b>Preliminaries</b>	<b>6</b>
2.1	Notation . . . . .	6
2.2	MPC in the Semi-honest Model . . . . .	6
2.3	Private Set Operation . . . . .	6
<b>3</b>	<b>Protocol Building Blocks</b>	<b>7</b>
3.1	Oblivious Transfer . . . . .	7
3.2	Multi-Query RPMT . . . . .	7
<b>4</b>	<b>Review of Pseudorandom Function</b>	<b>8</b>
4.1	Weak PRF from the DDH Assumption . . . . .	9
4.2	PRF from the DDH Assumption . . . . .	9
<b>5</b>	<b>The First Generic Construction of mqRPMT</b>	<b>10</b>
5.1	Definition of Commutative Weak PRF . . . . .	10
5.2	Construction of Commutative Weak PRF . . . . .	11
5.3	mqRPMT from Commutative Weak PRF . . . . .	11
<b>6</b>	<b>The Second Generic Construction of mqRPMT</b>	<b>14</b>
6.1	Definition of Permuted OPRF . . . . .	14
6.2	Construction of Permuted OPRF . . . . .	14
6.3	mqRPMT from Permuted OPRF . . . . .	16
<b>7</b>	<b>Connection Between mqRPMT and mqPMT</b>	<b>18</b>
7.1	Sigma-mqPMT . . . . .	18
7.2	mqRPMT* from Sigma-mqPMT . . . . .	19
<b>8</b>	<b>Applications of mqRPMT</b>	<b>20</b>
8.1	PSO Framework from mqRPMT . . . . .	20
8.2	Private-ID . . . . .	23
<b>9</b>	<b>Performance</b>	<b>24</b>
9.1	Implementation Details . . . . .	24
9.2	Experimental Setup . . . . .	24
9.3	Evaluation of mqRPMT . . . . .	24
9.4	Benchmark Comparison of PSO Protocols . . . . .	25
9.5	Tips For ECC-based Implementations . . . . .	27
<b>10</b>	<b>Summary and Perspective</b>	<b>28</b>
<b>A</b>	<b>Missing Definitions</b>	<b>32</b>
A.1	Weak Pseudorandom EGA . . . . .	32
<b>B</b>	<b>Instantiations of Sigma-mqPMT</b>	<b>32</b>
B.1	Sigma-mqPMT from DDH . . . . .	32
B.2	Sigma-mqPMT from FHE . . . . .	34
<b>C</b>	<b>Missing Security Proofs</b>	<b>34</b>
C.1	Proof of Permuted OPRF Based on the DDH Assumption . . . . .	34

# 1 Introduction

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

**Private set intersection.** PSI has found many applications such as privacy-preserving sharing, private contact discovery, DNA testing and pattern matching. 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<sup>+</sup>21, RS21] mainly rely on symmetric-key operations, except a little public-key operations 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.** 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<sup>+</sup>20, GMR<sup>+</sup>21]. For general-purpose PCSI (also known as circuit-PSI) [HEK12, PSTY19], the parties learn secret shares of the set intersection, which can be further fed into generic 2PC to compute  $g(X \cap Y)$  for arbitrary function  $g$ .

**Private set union.** Like PSI, PSU also has numerous applications in practice, such as cyber risk assessment and management via joint IP blacklists and joint vulnerability data. According to the underlying cryptographic techniques, existing PSU protocols can be roughly divided into two categories. The first is mainly based on public-key techniques [KS05, Fri07, HN10, DC17]. The second is mainly based on symmetric-key techniques [KRTW19, GMR<sup>+</sup>21, JSZ<sup>+</sup>22]. We refer to [ZCL<sup>+</sup>23] for a good survey of existing PSU protocols.

PSO protocols are primarily designed for balanced setting, in which the two sets' sizes are approximately the same. Recently, some works begin to consider unbalanced setting, in which one set is much more larger than the other. Among all PSO protocols, PSI has been extensively studied. In 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 unbalanced setting, a series of work [CLR17, CHLR18, CMdG<sup>+</sup>21] shows how to leverage fully homomorphic encryption (FHE) to build PSI protocols with sublinear complexity in the larger set size. In contrast to the affairs of PSI, the studies of PCSI and PSU are less satisfactory. As to PCSI, in balanced setting few protocols [PSTY19, IKN<sup>+</sup>20] achieve linear complexity, but the practical performance is poor. As pinpointed by [GMR<sup>+</sup>21], semi-honest PCSI – even in the simplest case, like PSI-card – is concretely about  $20\times$  slower and requires over  $30\times$  more communication than PSI. [CHLR18] also propose PSI-card and PSI-card-sum protocols based on generic 2PC in unbalanced setting, but these protocols are more of theoretical interest, and are not accompanied by implementations. As to 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<sup>+</sup>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 large constant in computation complexity, incurring a large efficiency gap compared with PSI: roughly  $25\times$  slower and requires at least  $3\times$  more communication than PSI.

It is somewhat surprising that different PSO protocols have significantly different efficiency. Why is this case? Observe that PSI essentially can be viewed as multi-query private membership test (mqPMT), which has very 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 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. Therefore, different approaches are employed for different PSOs, requiring much more engineering effort. We are motivated to seek for the minimal common protocol that enables all private set operations via a unified framework. Second, given the huge efficiency gap between PSI and other closely related protocols, we are also motivated to give efficient instantiations of the framework to close the gap. Last but not least, recall that the seminal PSI protocol known as DH-PSI [Mea86] (related ideas were appeared in [Sha80, HFH99]) is derived from the Diffie-Hellman key-exchange protocol based on the decisional Diffie-Hellman (DDH) assumption. After roughly four decades, DH-PSI is still the most easily understood and implemented one among numerous PSI protocols. Somewhat surprisingly, no counterpart is known in the PSU setting yet. It is curious to know if the DDH assumption can strike back. In summary, we are intrigued to know:

*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? Can the DDH assumption strike back with 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 itself already implies PSI-card; 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 in 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 propose 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 protocols.

**Connection to mqPMT.** mqRPMT is of great theoretical interest since it is the core building block of the PSO framework. It is thus interesting to investigate the relation between mqRPMT and mqPMT, 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. We argue that though mqRPMT\* deviates from standard mqRPMT in revealing additional information (intersection size) to the client, it could also be a desirable feature in application scenarios where both parties want to learn intersection size, for example, PSI-card-sum [IKN<sup>+</sup>20]. We leave the general connection between mqPMT and mqRPMT as a challenging open problem.

**Evaluations.** We give efficient instantiation of our generic framework from cwPRF-based mqRPMT protocol. We 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 (a.k.a. PSI) protocol generally is not applicable for computing PCSI and PSU. We examine the reverse direction, i.e., whether the core protocol under-

lying PSU can be used for computing PSI and PCSI. We identify that the recently emerged mqRPMT protocol [ZCL<sup>+</sup>23], which is a generalization of RPMT formalized in [KRTW19], is actually the central protocol underlying all the existing PSU protocols is actually mqRPMT. 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 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 instead. This subtle difference turns out to be significant. To see this, note that in mqRPMT the intersection information (a.k.a.  $x_i$  and  $e_i$ ) is shared between the 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. With mqRPMT in hand, PSI-card protocol immediately follows since the cardinality of intersection is exactly the Hamming weight of indication vector. PSI (resp. PSU) protocol can be done 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 show 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.** We 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 terminology introduced above, 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*, i.e., 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  $F : K \times D \rightarrow R$  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,  $\hat{F}$ .

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  $F$  via a common cryptographic hash function  $H$ , which will be modeled as a random oracle; (2)  $P_1$  computes and sends  $F_{k_1}(H(y))$  to  $P_2$ ; (3)  $P_2$  computes and sends  $F_{k_2}(H(x))$  and  $F_{k_2}(F_{k_1}(H(y)))$  to  $P_1$ ; (4)  $P_1$  then learns the test result by comparing  $F_{k_1}(F_{k_2}(H(x))) = ? F_{k_2}(F_{k_1}(H(y)))$ . The commutative property of  $F$  ensures the correctness. The weak pseudorandomness of  $F$  guarantees that  $P_2$  learns nothing and  $P_1$  learns nothing beyond the test result. In the above construction,  $F_{k_2}(F_{k_1}(H(\cdot))) = F_{k_1}(F_{k_2}(H(\cdot))) = \hat{F}_k(H(\cdot))$  serves as a pseudorandom encoding function in the joint view, while  $F_{k_1}(H(\cdot))$  and  $F_{k_2}(H(\cdot))$  serve as a partial encoding function in the individual views of 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  $F_{k_2}(F_{k_1}(H(y_i)))$  for each  $y_i \in Y$  in the same order of server's first move message  $F_{k_1}(H(y_i))$  does not lead to a secure  $(n_1, 1)$ -RPMT. The reason is that  $\{\hat{F}_k(H(y_i))\}_{i \in [n_1]}$  constitutes an order-preserving pseudorandom encoding of  $(y_1, \dots, y_{n_1})$ . 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  $(y_1, \dots, y_{n_1})$  independent of the order known by the server. A straightforward approach is to shuffle  $\{\hat{F}_k(H(y_i))\}$ . In this way, we obtain a  $(n_1, 1)$ -RPMT protocol from cwPRF, and can further batch it 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. We can further reduce the communication cost by inserting  $\{\hat{F}(H(y_i))\}$  into an order-hiding data structure such as Bloom filter, instead of shuffling them.

In Section 5.2 we show that cwPRF can be realized from DDH-like assumptions. Putting all the above together, DDH implies all PSO protocols. Notably, it strikes back with an incredibly simple PSU protocol. This once again demonstrates 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*, i.e., only one party say  $P_2$  is able to encode, and the other party say  $P_1$  can learn the codewords of its elements  $(y_1, \dots, y_{n_1})$  in a permuted order, while  $P_2$  learns nothing. 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 receiver) and  $P_2$  (acts as sender) engage in an OPRF protocol. Eventually,  $P_1$  obtains PRF values of  $(y_1, \dots, y_{n_1})$  as encodings, and  $P_2$  obtains a PRF key  $k$ . However, OPRF does not readily enable oblivious permuted encoding. The reason is that 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 sender additionally obtains a random permutation  $\pi$  over  $[n_1]$  besides PRF key  $k$ , while the receiver obtains PRF values in a permuted order as per  $\pi$ . pOPRF immediately implies a  $(n_1, 1)$ -RPMT protocol: The server with  $Y = (y_1, \dots, y_{n_1})$  and the client with  $X = \{x\}$  first engage in a pOPRF protocol. As a result, the server obtains  $\{F_k(y_{\pi(i)})\}_{i \in [n_1]}$ , while the client learns a PRF key  $k$  and a permutation  $\pi$ . The client then computes and sends  $F_k(x)$  to the server as RPMT query for  $x$ . Finally, the server learns if  $x \in Y$  by testing whether  $F_k(x) \in \{F_k(y_{\pi(i)})\}_{i \in [n_1]}$ , 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. Extending the above  $(n_1, 1)$ -RPMT to  $(n_1, n_2)$ -RPMT is straightforward by having the client reuse  $k$  and send  $\{F_k(x_i)\}_{i \in [n_2]}$  as RPMT queries.

The question 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<sup>1</sup>, then unmask the outputs. If the mask procedure is different per input, then the unmask procedure must be carried out accordingly. Therefore, OPRF protocols of this case cannot be easily adapted to pOPRF, since the receiver is unable to perform the unmask procedure over permuted masked outputs correctly, namely, 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 receiver might be able to unmask the permuted masked outputs correctly. Observe that the simplest way to perform unified masking is to apply a weak pseudorandom function  $F_s$  to the intermediate input  $H(x)$ , where  $H$  is a cryptographic hash function that maps input  $x$  to the domain of  $F_s$ . To enable efficient unmask procedure, 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 sender picks a random PRP key  $k$  for  $F$ , while the receiver with input  $X = (x_1, \dots, x_n)$  picks a random PRP key  $s$  for  $F$ . The receiver then sends  $\{F_s(H(x_i))\}_{i \in [n]}$  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.  $\{F_k(F_s(H(x_{\pi(i)})))\}_{i \in [n]}$ . Finally, the receiver applies  $F_s^{-1}$  to the permuted masked outputs, and will obtain  $\{F_k(H(x_{\pi(i)}))\}_{i \in [n]}$  by the commutative property.

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

**mqRPMT\* from Sigma-mqPMT.** Towards the goal of studying the connection between mqRPMT and mqPMT, we first abstract a category of mqPMT protocols called Sigma-mqPMT. The starting point of Sigma-mqPMT is Sigma-PMT. Roughly speaking, Sigma-PMT is a three-move protocol, which proceeds as below: (1) in the first move, the server holding a set  $Y$  sends a message  $a$  to the client, where  $a$  is best interpreted as an encoding of  $Y$ ; (2) in the second move, the client makes a test query  $q$  of its item  $x$ ; (3) in the last move, the server responds with  $z$ , and eventually the client can decide if  $x \in Y$  by running algorithm  $\text{Test}(a, q, x, z)$ . To enable efficient parallel composition, we introduce the following two properties for Sigma-PMT: (i) reusable property, which ensures the first move message can be safely reused over multi-instance; (ii) context-independent property, which means the test query only depends on the item in test. With these two properties, one can build mqPMT by running multiple instances of Sigma-PMT in parallel, without increasing round complexity. If the underlying Sigma-PMT

<sup>1</sup>Standard pseudorandomness denies input homomorphism. Rigorously speaking, we utilize the homomorphism over intermediate input.

additionally satisfies stateless testing, namely, Test algorithm can be done without learning  $(q, x)$ , we refer to the resulting mqPMT as Sigma-mqPMT, which captures the common form of several PSI protocols [Mea86, FIPR05, CLR17]. By utilizing the stateless property, we can tweak Sigma-mqPMT to permuted mqPMT via the “shuffle-then-test” approach, without incurring computation and communication overhead, while permuted mqPMT instantly implies mqRPMT\*. 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 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.

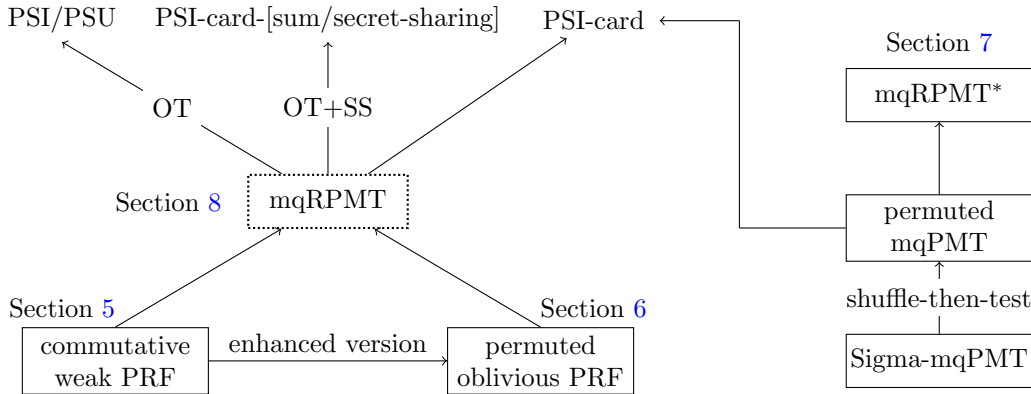


Figure 1: Summary of our main results. The rectangles with solid lines denote notions newly in this work. The rectangle with dotted lines denotes notion in previous work.

## 1.4 Related Works

We review previous PSI-card, PSI-card-sum and PSU protocols that are related to our work. Ion et al. [IKN<sup>+</sup>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 not efficient due to the usage of AHE, and as noted by the authors, detailed conversions to each category of protocols differ significantly, especially in the way of making use of the underlying AHE. In contrast to their work, we distill a broad class of PSI protocols as Sigma-mqPMT, then show how to tweak it to mqRPMT\* in a generic and black-box manner, without relying on any additional cryptographic tools. Our conversion works at a more abstract and lower level, and such generality lends it may find more potential applications. Miao et al. [MPR<sup>+</sup>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 can 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  $\lambda$  (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<sup>+</sup>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<sup>+</sup>21] differs from the original functionality defined in [IKN<sup>+</sup>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<sup>+</sup>21], the party without holding values is given the cardinality and sum of intersection. To highlight this subtle difference, we prefer to call the functionality presented in [GMR<sup>+</sup>21] as reverse PSI-card-sum.

**Concurrent work.** Very recently, Zhang et al. [ZCL<sup>+</sup>23] proposed a generic construction of mqRPMT with linear complexity from oblivious key-value store, set-membership encryption and oblivious vector decryption-then-test functionality. By instantiating their generic construction from public-key and symmetric-key encryption respectively and combining OT, they make the breakthrough by giving the first PSU protocol with optimal linear complexity. However, 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  $\kappa$  and  $\lambda$  to denote the computational and statistical parameter respectively. Let  $\mathbb{Z}_n$  be the set  $\{0, 1, \dots, n-1\}$ ,  $\mathbb{Z}_n^* = \{x \in \mathbb{Z}_n \mid \gcd(x, n) = 1\}$ . We use  $[n]$  to denote the set  $\{1, \dots, n\}$ , and use  $\text{Perm}[n]$  to denote all the permutations over the set  $\{1, \dots, n\}$ . We assume that every set  $X$  has a default order (e.g. lexicographical order), and write it as  $X = \{x_1, \dots, x_n\}$ . For a set  $X$ , we use  $|X|$  to denote its size and use  $x \xleftarrow{R} X$  to denote sampling  $x$  uniformly at random from  $X$ . We use  $(x_1, \dots, x_n)$  to denote a vector, and its  $i$ -th element is  $x_i$ . A function is negligible in  $\kappa$ , written  $\text{negl}(\kappa)$ , if it vanishes faster than the inverse of any polynomial in  $\kappa$ . 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  $\Pi$  be a two-party 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  $P_i$  where  $i \in \{1, 2\}$ , let  $\text{View}_{P_i}(x_1, x_2)$  denote the view of party  $P_i$  during an honest execution of  $\Pi$  on inputs  $x_1$  and  $x_2$ . The view consists of  $P_i$ 's input, random tape, and all messages exchanged as part of the  $\Pi$  protocol.

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

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

Roughly speaking, a protocol is secure if  $P_i$  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  $Y = \{y_1, \dots, y_{n_1}\}$  where  $y_i \in \{0, 1\}^\ell$ . The sender  $P_2$  inputs a set of elements  $X = \{x_1, \dots, x_{n_2}\}$  where  $x_i \in \{0, 1\}^\ell$  and possibly a set of values  $V = \{v_1, \dots, v_{n_2}\}$  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  $\mathcal{F}_{\text{PSO}}$  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 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 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  $\ell$ .

**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  $(b_1, \dots, b_n) \in \{0, 1\}^n$ .

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

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

### 3.2 Multi-Query RPMT

mqRPMT [ZCL+23] is a protocol where the client with input vector  $(x_1, \dots, x_n)$  interacts with a server holding a set  $Y$ . As a result, the server learns only the bit vector  $(e_1, \dots, e_n)$  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  $|X \cap Y|$ .

**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

**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  $Y = (y_1, \dots, y_{n_1})$ , 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  $x_i \in \{0, 1\}^\ell$ .

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

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

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 mix-match two characteristics, we obtain four types PMT protocols, shown in Table 1.

Table 1: The family of PMT protocols

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

mqPMT and PSI are the same protocol under different names. mqRPMT is formalized in [KRTW19, ZCL<sup>+</sup>23]. Permuted mqRPMT is introduced in [GMR<sup>+</sup>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 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:

- **Setup**( $1^\kappa$ ): on input a security parameter  $\kappa$ , outputs public parameter  $pp$ .  $pp$  specifies a family of keyed functions  $F : K \times D \rightarrow R$ , where  $K$  is the key space,  $D$  is domain, and  $R$  is range.
- **KeyGen**( $pp$ ): on input  $pp$ , outputs a secret key  $k \xleftarrow{R} K$ .
- **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:

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

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

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

*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  $a, b_i, c_i \xleftarrow{\mathbb{R}} \mathbb{Z}_p$ . 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  $\mathcal{O}_{\text{ror}}$ ,  $\mathcal{B}$  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  $\beta$ , where ‘0’ indicates real mode and ‘1’ indicates random mode.  $\mathcal{B}$  forwards  $\beta'$  to its own challenger.

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

*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  $k$ .

## 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  $H : D \rightarrow \mathbb{G}$ , then apply  $F_k$  in a cascade way,

<sup>2</sup>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  $\mathcal{O}_{\text{ror}}(1, \cdot)$  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.

yielding a composite function  $F_k \circ H : D \rightarrow \mathbb{G}$ . Assuming  $H$  is a random oracle, the pseudorandomness of the composite function  $F_k \circ H$  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.

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

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

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

*Proof.* We now reduce the pseudorandomness of  $F_k(H(\cdot))$  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  $pp = (\mathbb{G}, g, p, H)$  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  $H(x_i) := g^{b_i}$ .
- **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  $\beta$  to its own challenger.

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

*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  $F : K \times D \rightarrow R$ ,  $F$  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  $k' \xleftarrow{\mathbb{R}} K$ ,  $x \xleftarrow{\mathbb{R}} D$ , 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  $K \times D \rightarrow D$ .  $F$  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  $F : K \times D \rightarrow D$  and cryptographic hash function  $H : \{0, 1\}^\ell \rightarrow D$ .

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

**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  $k_1 \xleftarrow{R} K$ , then sends  $\{F_{k_1}(H(y_i))\}_{i \in [n_1]}$  to  $P_2$ .
2.  $P_2$  picks  $k_2 \xleftarrow{R} K$ , then computes and sends  $\{F_{k_2}(H(x_i))\}_{i \in [n_2]}$  to  $P_1$ .  $P_2$  also computes  $\{F_{k_2}(F_{k_1}(H(y_i)))\}_{i \in [n_1]}$ , picks a random permutation  $\pi \xleftarrow{R} [n_1]$ , then sends  $\{F_{k_2}(F_{k_1}(H(y_{\pi(i)})))\}_{i \in [n_1]}$  to  $P_1$ . An alternative choice instead of explicit shuffle is inserting  $\{F_{k_2}(F_{k_1}(H(y_i)))\}_{i \in [n_1]}$  to a Bloom filter, then sends the resulting filter to  $P_1$ . We slightly abuse the notation, and still use  $\Omega$  to denote the Bloom filter.
3.  $P_1$  computes  $\{F_{k_1}(F_{k_2}(H(x_i)))\}_{i \in [n_2]}$ , then sets  $e_i = 1$  iff  $F_{k_1}(F_{k_2}(H(x_i))) \in \Omega$ .

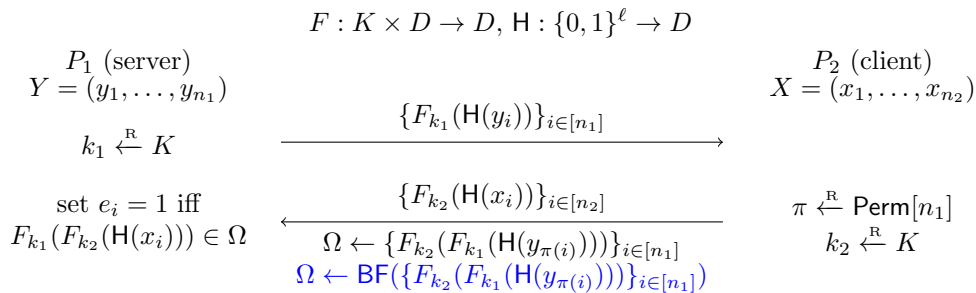


Figure 5: Multi-query RPMT from commutative weak PRF

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

**Correctness.** The above protocol is correct except the event  $E$  that  $F_{k_1}(F_{k_2}(\mathbf{H}(x))) = F_{k_1}(F_{k_2}(\mathbf{H}(y)))$  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  $\mathbf{H}(x) = \mathbf{H}(y)$ . By the collision resistance of  $\mathbf{H}$ , we have  $\Pr[E_0] = 2^{-\kappa}$ . Let  $E_1$  be the event that  $\mathbf{H}(x) \neq \mathbf{H}(y)$  but  $F_{k_1}(F_{k_2}(\mathbf{H}(x))) = F_{k_1}(F_{k_2}(\mathbf{H}(y)))$ , which can further be divided into sub-cases  $E_{10} - F_{k_2}(\mathbf{H}(x)) = F_{k_2}(\mathbf{H}(y))$  and  $E_{11} - F_{k_2}(\mathbf{H}(x)) \neq F_{k_2}(\mathbf{H}(y))$  but  $F_{k_1}(F_{k_2}(\mathbf{H}(x))) = F_{k_1}(F_{k_2}(\mathbf{H}(y)))$ . By the weak pseudo-randomness 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  $\kappa$ . Therefore, by union bound we have  $\Pr[E] \leq n_1 n_2 \cdot (\Pr[E_0] + \Pr[E_{10}] + \Pr[E_{11}]) = \text{negl}(\kappa)$ .

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

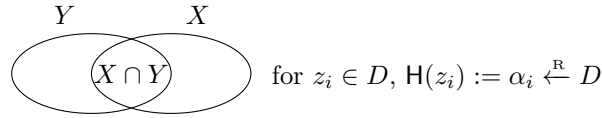
*Proof.* We exhibit simulators  $\text{Sim}_{P_1}$  and  $\text{Sim}_{P_2}$  for simulating corrupt  $P_1$  and  $P_2$  respectively, and argue the indistinguishability of the simulated transcript from the real execution. Let  $|X \cap Y| = m$ .

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

Hybrid<sub>0</sub>:  $P_2$ 's view in the real protocol.

Hybrid<sub>1</sub>: Given  $P_2$ 's input  $X$ ,  $\text{Sim}_{P_2}$  chooses the randomness for  $P_1$  (i.e., picks  $k_1 \xleftarrow{\text{R}} K$ ), and simulates with the knowledge of  $Y$ .

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



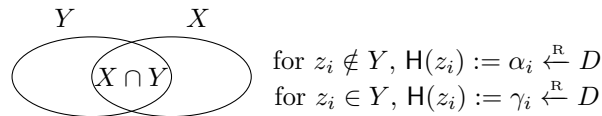
Clearly,  $\text{Sim}_{P_2}$ 's simulated view in Hybrid<sub>1</sub> is identical to  $P_2$ 's real view.

Hybrid<sub>2</sub>:  $\text{Sim}_{P_2}$  does not choose the randomness for  $P_1$  (i.e., picks  $k_1 \xleftarrow{\text{R}} K$ ), and simulates without the knowledge of  $Y$ . It emulates the random oracle  $\mathbf{H}$  honestly as before, and only changes the simulation of  $P_1$ 's message.

- $\text{Sim}_{P_2}$  outputs  $(\eta_1, \dots, \eta_{n_1})$  where  $\eta_i \xleftarrow{\text{R}} D$ .

We argue that the simulated view in Hybrid<sub>1</sub> and Hybrid<sub>2</sub> 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  $(\gamma_i, \eta_i)$  where  $\gamma_i \xleftarrow{\text{R}} D$ , and is asked to distinguish if  $\eta_i = F_k(\gamma_i)$  or  $\eta_i$  are random values.  $\mathcal{A}$  implicitly sets  $P_1$ 's randomness  $k_1 := k$ , and simulates as below.

- RO query: for each random oracle query  $\langle z_i \rangle$ , if  $z_i \notin Y$ , picks  $\alpha_i \xleftarrow{\text{R}} D$  and sets  $\mathbf{H}(z_i) := \alpha_i$ ; if  $z_i \in Y$ , sets  $\mathbf{H}(z_i) := \gamma_i$ .
- outputs  $(\eta_1, \dots, \eta_{n_1})$ .



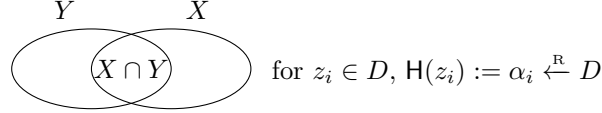
If  $\eta_i = F_k(\gamma_i)$  for  $i \in [n_1]$ , then  $\mathcal{A}$ 's simulation is identical to Hybrid<sub>1</sub>. If  $\eta_i$  are random values, then  $\mathcal{A}$ 's simulation is identical to Hybrid<sub>2</sub>.

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

Hybrid<sub>0</sub>:  $P_1$ 's view in the real protocol.

Hybrid<sub>1</sub>: Given  $P_1$ 's input  $Y$  and output  $(e_1, \dots, e_{n_1})$ ,  $\text{Sim}_{P_1}$  chooses the randomness for  $P_2$  (i.e., picks  $k_2 \xleftarrow{\text{R}} K$  and a random permutation  $\pi$  over  $[n_1]$ ), and simulates with the knowledge of  $X$ .

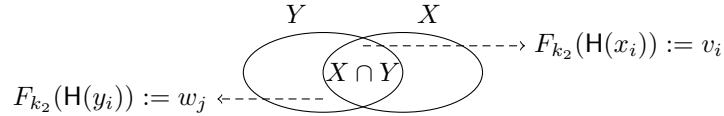
- RO queries:  $\text{Sim}_{P_1}$  emulates the random oracle  $\text{H}$  honestly. For each query  $\langle z_i \rangle$ ,  $\text{Sim}_{P_1}$  picks  $\alpha_i \xleftarrow{\text{R}} D$  and assigns  $\text{H}(z_i) := \alpha_i$ .
- $\text{Sim}_{P_1}$  outputs  $\{F_{k_2}(\text{H}(x_i))\}_{i \in [n_1]}$  and  $\Omega \leftarrow \{F_{k_2}(F_{k_1}(\text{H}(y_{\pi(i)}))\}_{i \in [n_1]}$ .



Clearly,  $\text{Sim}_{P_1}$ 's simulation in Hybrid<sub>1</sub> is identical to the real view of  $P_1$ .

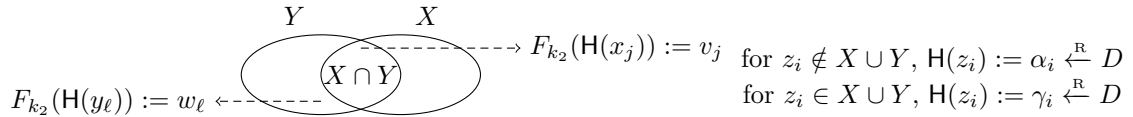
Hybrid<sub>2</sub>:  $\text{Sim}_{P_1}$  does not choose randomness for  $P_2$ , and simulates without the knowledge of  $X$ . It simulates the random oracle  $\text{H}$  honestly as before, and changes its simulation of  $P_2$ 's message. Let  $m$  be the Hamming weight of  $(e_1, \dots, e_{n_1})$ .

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



We argue that the view in Hybrid<sub>1</sub> and Hybrid<sub>2</sub> 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  $(\gamma_i, \eta_i)$  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  $\text{H}(z_i) := \alpha_i$ ; for  $z_i \in X \cup Y$ , assigns  $\text{H}(z_i) := \gamma_i$ .
- For each  $z_i \in X$ ,  $\mathcal{A}$  picks out the associated  $\eta_i$  to form  $\{v_j\}_{j \in [n_2]}$ ; for each  $z_i \in Y - X \cap Y$ ,  $\mathcal{A}$  picks out the associated  $\eta_i$  to form  $\{w_\ell\}_{\ell \in [n_1 - m]}$ . Finally,  $\mathcal{A}$  outputs  $\{v_j\}_{j \in [n_2]}$  and a random permutation of  $(\{F_{k_1}(v_j)\}_{x_j \in X \cap Y}, \{F_{k_1}(w_\ell)\}_{\ell \in [n_1 - m]})$ .



If  $\eta_i = F_k(\gamma_i)$ , then  $\mathcal{A}$ 's simulation is identical to Hybrid<sub>1</sub>. If  $\eta_i$  are random values, then  $\mathcal{A}$ 's simulation is identical to Hybrid<sub>2</sub>.

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  $\{F_{k_2}(F_{k_1}(\mathbf{H}(y_i)))\}_{i \in [n_1]}$  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  $n_1$  group elements to  $1.44\lambda \cdot n_1$  bits (with false positive probability  $2^{-\lambda}$ ), where  $\lambda$  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<sup>3</sup>. To the best of knowledge, Bloom filter merely serves as a space-efficient data structure in previous works [KLS<sup>+</sup>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 sender learns a PRF key  $k$  and the receiver learns  $F_k(x_1), \dots, F_k(x_n)$ , where  $F$  is a pseudorandom function (PRF) and  $(x_1, \dots, x_n)$  are the receiver's inputs. Nothing about the receiver's inputs is revealed to the sender and nothing more about the key  $k$  is revealed to the receiver.

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

**Parameters:** Number of OPRF queries  $n$ .

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

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

Figure 6: Ideal functionality  $\mathcal{F}_{\text{pOPRF}}$  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  $\mathbf{H}(x)$ , 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.

*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  $\mathbf{H}$  is a random oracle and  $F$  is a family of cwPRPs.*

*Proof.* We exhibit simulators  $\text{Sim}_{P_1}$  and  $\text{Sim}_{P_2}$  for simulating corrupt  $P_1$  and  $P_2$  respectively, and argue the indistinguishability of produced transcript from the real execution.

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

Hybrid<sub>0</sub>:  $P_1$ 's view in the real protocol.

<sup>3</sup>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  $\{h_1(x), \dots, h_k(x)\}$  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 not meet order-hiding property.



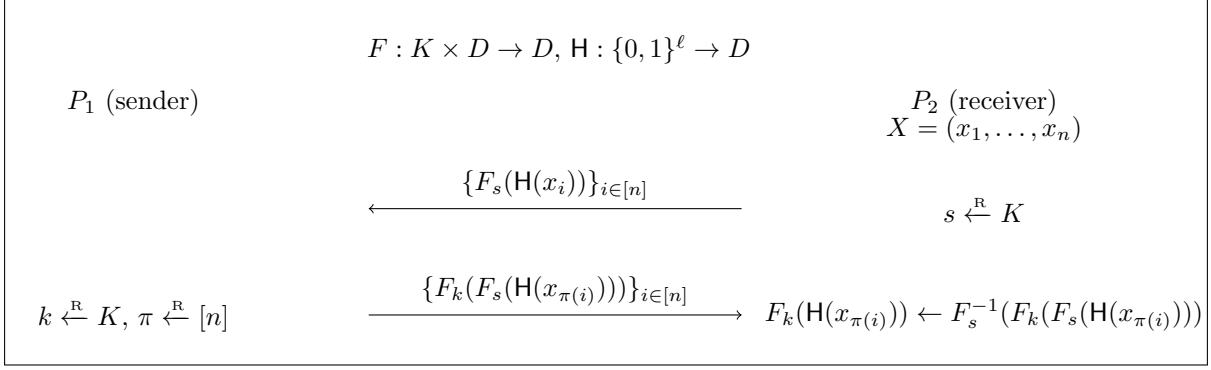


Figure 7: Permuted OPRF from cwPRP

Hybrid<sub>1</sub>: Given  $P_1$ 's output  $k$  and  $\pi$ ,  $\text{Sim}_{P_1}$  chooses the randomness  $s$  for  $P_2$ , and simulates with the knowledge of  $X = (x_1, \dots, x_n)$ :

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

Clearly,  $\text{Sim}_{P_1}$ 's simulated view in Hybrid<sub>1</sub> is identical to  $P_1$ 's real view.

Hybrid<sub>2</sub>:  $\text{Sim}_{P_1}$  does not choose the randomness for  $P_2$ , and simulates without the knowledge of  $X$ . It honestly emulates random oracle  $H$  as in Hybrid<sub>1</sub>, and only changes the simulation of  $P_2$ 's message.

- $\text{Sim}_{P_1}$  outputs  $(\eta_1, \dots, \eta_n)$  where  $\eta_i \xleftarrow{R} D$ .

We argue that the view in Hybrid<sub>1</sub> and Hybrid<sub>2</sub> are computationally indistinguishable. Let  $\mathcal{A}$  be a PPT adversary against the weak pseudorandom of  $F_s$ . Given a real-or-random oracle  $\mathcal{O}_{\text{ror}}(\cdot)$ ,  $\mathcal{A}$  is asked to distinguish which mode he is in.  $\mathcal{A}$  queries the  $\mathcal{O}_{\text{ror}}(\cdot)$   $n$  times, and obtains  $(\gamma_i, \eta_i)$  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  $H(z_i) := \alpha_i$ ; if  $z_i \in X$ , assigns  $H(x_i) := \gamma_i$ .
- Outputs  $(\eta_1, \dots, \eta_n)$ .

Clearly, if  $\eta_i = F_s(\gamma_i)$ ,  $\mathcal{A}$  simulates Hybrid<sub>1</sub>. Else, it simulates Hybrid<sub>2</sub>. Thereby,  $\text{Sim}_{P_1}$ 's simulated view is computationally indistinguishable to  $P_1$ 's real view.

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

Hybrid<sub>0</sub>:  $P_2$ 's view in the real protocol.

Hybrid<sub>1</sub>: Given  $P_2$ 's input  $X = (x_1, \dots, x_n)$  and output  $\{F_k(H(x_{\pi(i)}))\}_{i \in [n]}$ ,  $\text{Sim}_{P_2}$  emulates the random oracle  $H$  honestly, picks  $s \xleftarrow{R} \mathbb{Z}_p$ , simulates message from  $P_1$  as  $\{F_s(F_k(H(x_{\pi(i)})))\}_{i \in [n]}$ .

According to the commutative property of cwPRF,  $\text{Sim}_{P_2}$ 's simulated view is identical to the real view.

This proves the theorem.  $\square$

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  $H : \{0, 1\}^\ell \rightarrow \mathbb{G}$ .

**Inputs:** The receiver  $P_2$  inputs a set  $X = \{x_1, \dots, x_n\}$ , where  $x_i \in \{0, 1\}^\ell$ .

**Protocol:**

1.  $P_2$  picks  $s \xleftarrow{R} \mathbb{Z}_p$ , then sends  $(H(x_1)^s, \dots, H(x_n)^s)$  to the sender  $P_1$ .
2.  $P_1$  picks  $k \xleftarrow{R} \mathbb{Z}_p$  and computes  $(H(x_1)^{sk}, \dots, H(x_n)^{sk})$ , then picks a random permutation  $\pi$  over  $[n]$  and sends  $y'_i = H(x_{\pi(i)})^{sk}$  for  $i \in [n]$  to  $P_2$ .
3.  $P_1$  outputs  $k$  and  $\pi$ .
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  $F : K \times D \rightarrow R$ . 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:** The server  $P_1$ 's set size  $n_1$  and the client  $P_2$ 's set size  $n_2$ , a permuted OPRF for  $F : K \times D \rightarrow R$ .

**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  $P_2$  invoke the permuted OPRF protocol. At the end of the protocol,  $P_1$  obtains  $\{F_k(y_{\pi(i)})\}_{i \in [n_1]}$ ,  $P_2$  obtains  $k$  and a permutation  $\pi$  over  $[n_1]$ .
2.  $P_2$  computes and sends  $(F_k(x_1), \dots, F_k(x_{n_2}))$  to  $P_1$ .
3.  $P_1$  sets  $e_i = 1$  iff  $F_k(x_i) \in \{F_k(y_{\pi(i)})\}_{i \in [n_1]}$ .

$$F : K \times D \rightarrow R$$

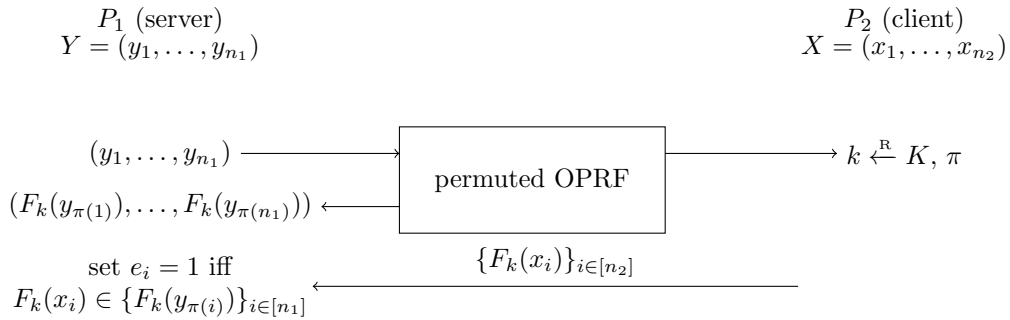


Figure 9: mqRPMT from permuted OPRF

**Correctness.** The above protocol is correct except the case  $E = \bigvee_{i,j} E_{ij}$  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



- query the real-or-random oracle with  $(y_{\pi(1)}, \dots, y_{\pi(n_1)})$  and obtain  $(\zeta_1, \dots, \zeta_{n_1})$  in return; then generate its stage 1's view by invoking the simulator for  $P_1$  of permuted OPRF with input  $(y_1, \dots, y_{n_1})$  and output  $(\zeta_1, \dots, \zeta_{n_1})$ .

Clearly, if  $\mathcal{A}$  queries the real oracle, then its simulation is identical to Hybrid<sub>1</sub>. Else, its simulation is identical to that Hybrid<sub>2</sub>. This reduces the computational indistinguishability of views in Hybrid<sub>1</sub> and Hybrid<sub>2</sub> to the pseudorandomness of  $F_k(\cdot)$ . Therefore,  $\text{Sim}_{P_1}$ '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 multiplications,  $n_2$  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  $\text{Test}(a, x, q, z)$ . 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  $(q_i, z_i)$ ,  $\text{Test}(a, x_i, q_i, z_i)$  can work in a memoryless way, namely, without looking at  $(x_i, q_i)$ . In this case, the test algorithm can be simplified as  $\text{Test}(a, z_i)$ .

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.

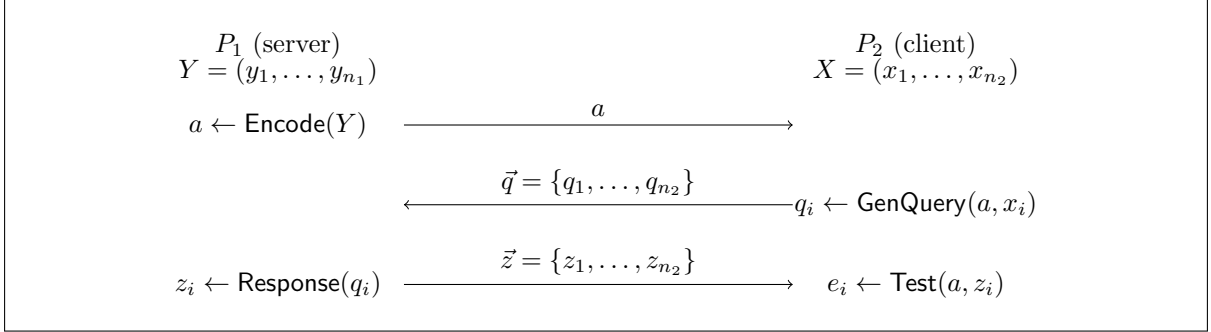


Figure 10: Sigma-mqPMT

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

- **Locality:**  $z_i \approx \text{Sim}'(e_i; r_i)$ , a.k.a. the  $i$ -th response can be emulated via invoking a sub-routine  $\text{Sim}'(e_i)$  with independent random coins  $r_i$ ;
- **Order invariance:**  $a \approx \text{Sim}''(\{e_{\pi(i)}, r_{\pi(i)}\}_{i \in [n_2]}; s)$ , where  $\pi$  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  $\text{Sim}_{P_1}$  and  $\text{Sim}_{P_2}$  for simulating corrupt server  $P_1$  and corrupt client  $P_2$  respectively. Let  $|X \cap Y| = m$ .

**Security against corrupt client.**  $\text{Sim}_{P_2}$  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  $\text{Sim}_{P_2}$  is indistinguishable from the real execution. We formally show  $\text{Sim}_{P_2}$ 's simulation is indistinguishable from the real execution via a sequence of hybrid transcripts.

Hybrid<sub>0</sub>:  $P_2$ 's view in the real protocol.

Hybrid<sub>1</sub>:  $\text{Sim}_{P_2}$  chooses the randomness for  $P_1$ , and simulates with the knowledge of  $Y$ . Clearly,  $\text{Sim}_{P_2}$ 's simulation is identical to the real view of  $P_2$ .

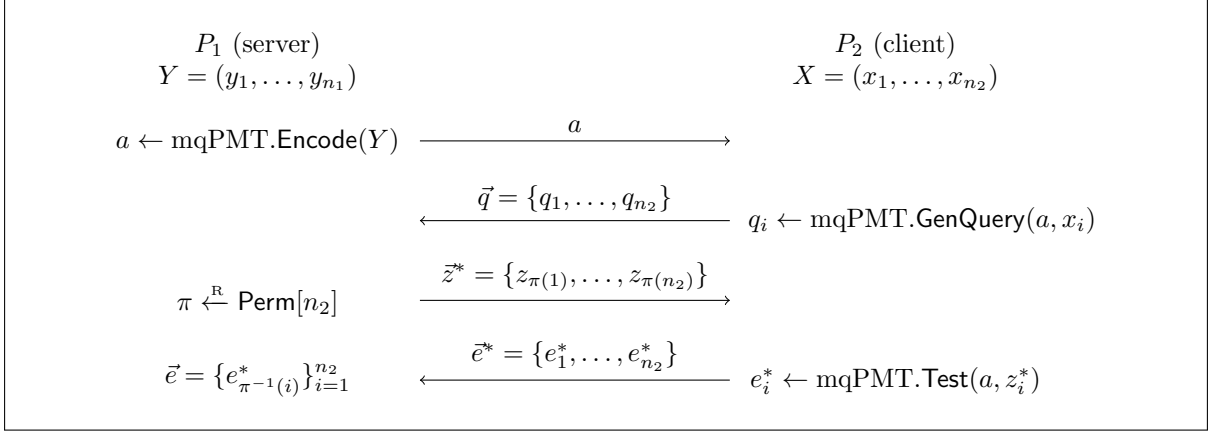


Figure 11: mqRPMT\* from Sigma-mqPMT

Hybrid<sub>2</sub>:  $\text{Sim}_{P_2}$  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  $\text{Sim}'(e_i^*; r_i) \rightarrow z_i^*$ , obtaining  $\vec{z}^* = (z_1^*, \dots, z_n^*)$ .
- run  $\text{Sim}''(\{(e_i^*, r_i)\}_{i \in [n_2]}; s) \rightarrow a$ .

By the *locality* and *order invariance* properties, the simulated view in Hybrid<sub>2</sub> and Hybrid<sub>1</sub> are computationally indistinguishable based on semi-honest security of mqPMT on  $P_2$  side.

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

Hybrid<sub>0</sub>:  $P_1$ 's view in the real protocol.

Hybrid<sub>1</sub>:  $\text{Sim}_{P_1}$  chooses the randomness for  $P_2$ , and simulates with the knowledge of  $X$ . Clearly,  $\text{Sim}_{P_1}$ 's simulation is identical to the real view of  $P_1$ .

Hybrid<sub>2</sub>:  $\text{Sim}_{P_1}$  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  $\pi$  over  $[n_2]$  and computes  $\vec{e}^* = \pi^{-1}(\vec{e})$ , outputs  $(\vec{q}, \vec{e}^*)$ .

Clearly, the view in Hybrid<sub>1</sub> and Hybrid<sub>2</sub> 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.*

**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  $Y = \{y_1, \dots, y_{n_1}\}$ , where  $y_i \in \{0, 1\}^\ell$ . The sender  $P_2$  inputs a set  $X = \{x_1, \dots, x_{n_2}\}$  and  $V = \{v_1, \dots, v_{n_2}\}$ , 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  $(x_1, \dots, x_{n_2})$  and  $(v_1, \dots, v_{n_2})$  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  $X = \{x_1, \dots, x_{n_2}\}$  invoke  $\mathcal{F}_{\text{mqRPMT}}$ .  $P_1$  obtains an indication bit vector  $\vec{e} = (e_1, \dots, e_{n_2})$ .  $P_2$  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  $\mathcal{F}_{\text{OT}}$ .  $P_1$  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  $\{x_i \mid e_i = 0\}_{i \in [n_2]} = X \setminus Y$ , and outputs  $\{X \setminus Y\} \cup Y = X \cup Y$ .
  - **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  $S' = \{\sum_{i=1}^{n_2} v_i \mid e_i = 1\}_{i \in [n_2]} + \{\sum_{i=1}^{n_2} r_i\}_{i \in [n_2]} \bmod q$ , then sends  $S'$  and the Hamming weight of  $\vec{e}$  to  $P_2$ .  $P_2$  computes  $S = (S' - r')$  mod  $q$ .
  - **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

*Proof.* We exhibit simulators  $\text{Sim}_{P_1}$  and  $\text{Sim}_{P_2}$  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.**  $\text{Sim}_{P_2}$  simulates the view of corrupt sender  $P_2$ , which consists of  $P_2$ 's randomness, input, output and received messages. We formally show  $\text{Sim}_{P_2}$ 's simulation is indistinguishable from the real execution via a sequence of hybrid transcripts.

Hybrid<sub>0</sub>:  $P_2$ '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<sub>1</sub>:  $\text{Sim}_{P_2}$  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.**  $\text{Sim}_{P_1}$  simulates the view of corrupt receiver  $P_1$ , which consists of  $P_1$ 's randomness, input, output and received messages. We formally show  $\text{Sim}_{P_1}$ 's simulation is indistinguishable from the real execution via a sequence of hybrid transcripts.

Hybrid<sub>0</sub>:  $P_1$ '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) and the OT part of view (stage 2).

Hybrid<sub>1</sub>: Given  $P_1$ 's input  $Y = (y_1, \dots, y_{n_1})$  and output  $X \cup Y$ ,  $\text{Sim}_{P_1}$  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  $X \setminus Y$ , 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. □

We compare our PSI-card protocol with closely related protocols. Huberman et al. [HFH99] proposed the first PSI-card protocol but did not provide 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 of 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<sup>+</sup>20, GMR<sup>+</sup>21] as below. As mentioned in the introduction part, the PSI-card-sum protocols presented in [IKN<sup>+</sup>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<sup>+</sup>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<sup>+</sup>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.

## 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 on the input distribution. Looking ahead, such relaxation may lead to nice efficiency improvement.

In this work, we instantiate the generic private-ID construction as below: (1) realize the distributed OPRF protocol by running the multi-point OPRF [CM20] 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  $G : K \times D \rightarrow R$ , 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 build upon the OpenSSL library [Opea], 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<sup>+</sup>21], whose item length is set as 61 bits in default and cannot exceed 64 bits.
- The PSI-card-sum protocol [IKN<sup>+</sup>20] and the private-id protocol [BKM<sup>+</sup>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 relies on special elliptic curve `Curve25519` realized in the highly-optimized Dalek library. 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  $\blacklozenge$  and  $\blacktriangledown$  respectively. For protocols based on `Curve25519`, we denote with  $\blackstar$ .

### 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			
<b>Speedup</b>		1.6-2.3 $\times$	1.9-3.0 $\times$	1.8-2.8 $\times$	1.2-1.3 $\times$	1.5-1.9 $\times$	1.6-2.2 $\times$	–	–	–
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			
<b>Speedup</b>		1.6-2.2 $\times$	1.6-2.3 $\times$	1.6-2.2 $\times$	1.1-1.3 $\times$	1.4-1.8 $\times$	1.5-2 $\times$	–	–	–
mqRPMT $\blackstar$	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			
<b>Speedup</b>		1.7-2.6 $\times$	2.0-3.3 $\times$	1.9-3.6 $\times$	1.1-1.1 $\times$	1.4-1.8 $\times$	1.7-2.4 $\times$	–	–	–

## 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<sup>+</sup>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 with the state-of-the-art PSI protocol. We include them merely for illustrative purposes. PSI protocols from public-key techniques are thought to be very inefficient. Our experiment demonstrates that by carefully choosing modern crypto library with optimized parameters they could be pretty practical. Our mqRPMT-based PSI protocol is more than an order of magnitude faster than the DH-PSI protocol<sup>4</sup> implemented in [PRTY19]. By leveraging the features of Curve25519 in important ways (see Section 9.5 in details), our re-implemented DH-PSI protocol (denoted by DH-PSI $\blackstar$ ) achieves 26 $\times$  speedup, which is arguably the most efficient implementation known to date.

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] $\blackstar$	5.51	88.64	1418.20	5.82	90.79	1498.67	0.30	4.74	76.60
Our PSI $\blacklozenge$	<b>0.50</b>	<b>7.24</b>	<b>114.66</b>	<b>1.71</b>	<b>10.50</b>	<b>142.45</b>	<b>0.67</b>	<b>10.38</b>	<b>165.77</b>
Our PSI $\blacktriangledown$	<b>0.55</b>	<b>8.04</b>	<b>128.18</b>	<b>1.73</b>	<b>11.02</b>	<b>148.18</b>	<b>0.41</b>	<b>6.38</b>	<b>101.63</b>
Our PSI $\blackstar$	<b>0.29</b>	<b>3.56</b>	<b>55.11</b>	<b>1.19</b>	<b>6.38</b>	<b>75.56</b>	<b>0.40</b>	<b>6.25</b>	<b>99.71</b>
DH-PSI $\blackstar$	<b>0.22</b>	<b>3.39</b>	<b>54.79</b>	<b>0.92</b>	<b>5.57</b>	<b>69.31</b>	<b>0.28</b>	<b>4.57</b>	<b>74.1</b>

Recently, Rosulek and Trieu [RT21] propose 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

<sup>4</sup>We remark that except inefficiency, their implementation also has a severe security issue. More precisely, they realize the hash-to-point function  $\{0,1\}^* \rightarrow \mathbb{G}$  as  $x \mapsto g^{\mathbf{H}(x)}$ , where  $\mathbf{H}$  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.

surprisingly, Table 4 shows that for small sets our mQRPM-based PSI protocol is faster than their protocol in LAN setting, and our re-implemented DH-PSI is faster than their protocol in all settings.

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] <sup>★</sup>	50.0	71.0	147.3	224.1	260.2	457.9	17.9	34.1	66.3
Our PSI <sup>★</sup>	<b>41.9</b>	<b>69.5</b>	<b>99.3</b>	<b>577.0</b>	<b>582.9</b>	<b>646.1</b>	<b>38.6</b>	<b>63.5</b>	<b>113.3</b>
DH-PSI <sup>★</sup>	<b>16.49</b>	<b>31.80</b>	<b>56.91</b>	<b>210.42</b>	<b>227.33</b>	<b>252.32</b>	<b>18.48</b>	<b>36.68</b>	<b>72.8</b>

**PSI-card.** We compare our mQRPM-based PSI-card protocol to the PSI-card protocol in [GMR<sup>+</sup>21]. Table 5 shows that our protocol achieves a 2.4 – 10.5× speedup in running time, and reduces the communication cost by a factor 10.9 – 14.8×.

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 <sup>+</sup> 21]	1.00	8.41	126.01	8.60	27.46	323.52	2.93	55.49	1030
Our PSI-card <sup>♦</sup>	<b>0.49</b>	<b>7.20</b>	<b>114.31</b>	<b>1.30</b>	<b>9.68</b>	<b>136.06</b>	<b>0.52</b>	<b>8.36</b>	<b>133.71</b>
Our PSI-card <sup>▼</sup>	<b>0.53</b>	<b>8.00</b>	<b>128.00</b>	<b>1.35</b>	<b>10.16</b>	<b>141.31</b>	<b>0.27</b>	<b>4.35</b>	<b>69.6</b>
Our PSI-card <sup>★</sup>	<b>0.27</b>	<b>3.51</b>	<b>54.89</b>	<b>0.82</b>	<b>5.42</b>	<b>68.31</b>	<b>0.26</b>	<b>4.23</b>	<b>67.70</b>

**PSI-card-sum.** We compare our mQRPM-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<sup>+</sup>20]. We do not compare the protocol described in [GMR<sup>+</sup>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<sup>+</sup>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<sup>+</sup>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<sup>+</sup>20], our protocol is roughly 28.5 – 76.3× faster and reduces the communication cost by a factor 7.4×.

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 <sup>+</sup> 20] <sup>▼</sup> (deployed)	23.64	176.34	–	30.10	186.29	–	2.72	43.24	–
Our PSI-card-sum <sup>♦</sup>	<b>0.51</b>	<b>7.22</b>	<b>113.66</b>	<b>1.46</b>	<b>9.68</b>	<b>136.27</b>	<b>0.64</b>	<b>9.89</b>	<b>157.80</b>
Our PSI-card-sum <sup>▼</sup>	<b>0.57</b>	<b>8.12</b>	<b>129.66</b>	<b>1.94</b>	<b>11.83</b>	<b>157.66</b>	<b>0.38</b>	<b>5.87</b>	<b>93.74</b>
Our PSI-card-sum <sup>★</sup>	<b>0.31</b>	<b>3.73</b>	<b>57.44</b>	<b>1.36</b>	<b>6.53</b>	<b>76.16</b>	<b>0.37</b>	<b>5.75</b>	<b>91.70</b>

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<sup>+</sup>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], which has not been fixed yet. The corresponding cells are marked with “–”.

**PSU.** We compare our mQRPM-based PSU protocol to the state-of-the-art PSU protocols in [GMR<sup>+</sup>21, ZCL<sup>+</sup>23, JSZ<sup>+</sup>22]. The work [ZCL<sup>+</sup>23] provides two PSU protocols from public-key and symmetric-key

respectively. The work [JSZ+22] also provides two PSU protocols called PSU-R and PSU-R. We choose the most efficient PKE-PSU [ZCL+23] and PSU-S [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 is  $2\times$  smaller in terms of communication cost, and achieves a  $2.7 - 17\times$  speedup.

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] <sup>♦</sup>	4.87	12.19	141.38	5.78	15.75	182.88	1.35	21.41	342.38
[ZCL+23] <sup>▼</sup>	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 <sup>♦</sup>	<b>0.52</b>	<b>7.27</b>	<b>114.44</b>	<b>1.70</b>	<b>10.56</b>	<b>143.29</b>	<b>0.68</b>	<b>10.38</b>	<b>165.77</b>
Our PSU <sup>▼</sup>	<b>0.57</b>	<b>8.04</b>	<b>128.20</b>	<b>1.76</b>	<b>10.92</b>	<b>148.15</b>	<b>0.41</b>	<b>6.38</b>	<b>101.63</b>
Our PSU <sup>★</sup>	<b>0.30</b>	<b>3.55</b>	<b>55.48</b>	<b>1.19</b>	<b>6.38</b>	<b>74.96</b>	<b>0.40</b>	<b>6.25</b>	<b>99.71</b>

**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]. The experimental results in Table 8 show that our private-ID protocol achieves a  $2.7 - 4.9\times$  speedup comparing to the existing most computation efficient private-ID protocol of [GMR+21], while its bandwidth is only marginally larger than the most communication efficient private-ID protocol [BKM+20]. Thereby, our protocol is arguably the most efficient one in terms of monetary cost. By instantiating the distributed OPRF from the current best OPRF due to [RR22], we will obtain a private-ID protocol with better performance.

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] <sup>★</sup>	2.21	37.56	671.75	7.98	46.97	710.94	<b>1.00</b>	<b>15.97</b>	<b>226.70</b>
Our Private-ID <sup>♦</sup>	<b>0.77</b>	<b>8.40</b>	<b>114.45</b>	<b>2.91</b>	<b>13.62</b>	<b>148.48</b>	<b>1.46</b>	<b>20.52</b>	<b>330.40</b>
Our Private-ID <sup>▼</sup>	<b>0.89</b>	<b>9.57</b>	<b>146.73</b>	<b>3.07</b>	<b>14.53</b>	<b>186.98</b>	<b>1.13</b>	<b>16.43</b>	<b>266.31</b>
Our Private-ID <sup>★</sup>	<b>0.61</b>	<b>5.11</b>	<b>71.57</b>	<b>2.83</b>	<b>10.06</b>	<b>114.66</b>	<b>1.13</b>	<b>16.31</b>	<b>265.15</b>

## 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 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 avoid if possible.

Regarding the two caveats discussed above, the following questions arise: (1) it is 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 circumstance.

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 our cwPRF-based mqRPMT protocol based on `Curve25519` over that based on NIST P-256. To the best of our knowledge, this is also the first time that `Curve25519` fully unleashes its advantages in the area of private set operations. Prior to this work, Rosulek and Tereu [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) does not apply to their construction because it requires encoding/decoding EC points to bit strings (thus 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 rushy 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 demonstrates 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

In this work, we show that mqRPMT protocol is complete for most private set operations. By coupling with OT, we create a unified PSO framework from mqRPMT, which greatly reduces the deployment and maintenance costs of PSO in the real world. We present two generic constructions of mqRPMT from newly introduced cryptographic primitives. Instantiations from the DDH assumption yield concrete

mqRPMT protocols and thus all their enabling PSO protocols with optimal asymptotic complexity and good practical efficiency. To sum up, we view the PSO framework from mqRPMT as well as its efficient construction and implementation as the main contribution of this work.

Our PSO framework is rather attractive given its simplicity. It allows us to interpret various PSO protocols through the lens of mqRPMT, and yields a family of concrete PSO protocols that are competitive or superior to existing ones. Along the way of building mqRPMT, we introduce cwPRF and pOPRF. The notion of 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 present a semi-generic conversion from a category mqPMT protocols called Sigma-mqPMT to mqRPMT, making the first step towards investigating relations between the two core protocols. As an application of such conversion, we obtain a mqRPMT protocol from FHE which is suitable in the unbalanced setting. However, the resulting mqRPMT is a slightly weak version in the sense that the intersection size is leaked to the sender. We left the construction of standard mqRPMT in the unbalanced setting as an open problem.

To demonstrate the efficiency of our framework, we open-source our C++ implementation with detailed documentations. When conducting performance comparison, we find that quite a few 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 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<sup>+</sup>20] Prasad Buddharapu, Andrew Knox, Payman Mohassel, Shubho Sengupta, Erik Taubeneck, and Vlad Vlaskin. Private matching for compute. 2020. <https://eprint.iacr.org/2020/599>.
- [BLS01] Dan Boneh, Ben Lynn, and Hovav Shacham. Short signatures from the weil pairing. In *Advances in Cryptology - ASIACRYPT 2001*, volume 2248 of *LNCS*, pages 514–532, 2001.
- [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<sup>+</sup>21] Kelong Cong, Radames Cruz Moreno, Mariana Botelho da Gama, Wei Dai, Iliia Iliashenko, Kim Laine, and Michael Rosenberg. Labeled PSI from homomorphic encryption with reduced computation and communication. In *CCS '21: 2021 ACM SIGSAC Conference on Computer and Communications Security*, pages 1135–1150. ACM, 2021.
- [DC17] Alex Davidson and Carlos Cid. An efficient toolkit for computing private set operations. In *Information Security and Privacy - 22nd Australasian Conference, ACISP 2017*, volume 10343 of *Lecture Notes in Computer Science*, pages 261–278. Springer, 2017.
- [DCW13] Changyu Dong, Liqun Chen, and Zikai Wen. When private set intersection meets big data: an efficient and scalable protocol. In *CCS 2013*, pages 789–800, 2013.
- [FIPR05] Michael J. Freedman, Yuval Ishai, Benny Pinkas, and Omer Reingold. Keyword search and oblivious pseudorandom functions. In *Theory of Cryptography, Second Theory of Cryptography Conference, TCC 2005*, volume 3378 of *Lecture Notes in Computer Science*, pages 303–324. Springer, 2005.
- [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<sup>+</sup>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<sup>+</sup>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.
- [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<sup>+</sup>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<sup>+</sup>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<sup>+</sup>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.
- [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<sup>+</sup>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 pseudo-random functions. In *36th Annual Symposium on Foundations of Computer Science, FOCS 1995*, pages 170–181. IEEE Computer Society, 1995.
- [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>.
- [ZCL<sup>+</sup>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  $\star : \mathbb{G} \times X \rightarrow X$  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  $\{\phi_g : X \rightarrow X\}_{g \in G}$  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  $x_i, u_i \xleftarrow{R} X$ .

## 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).

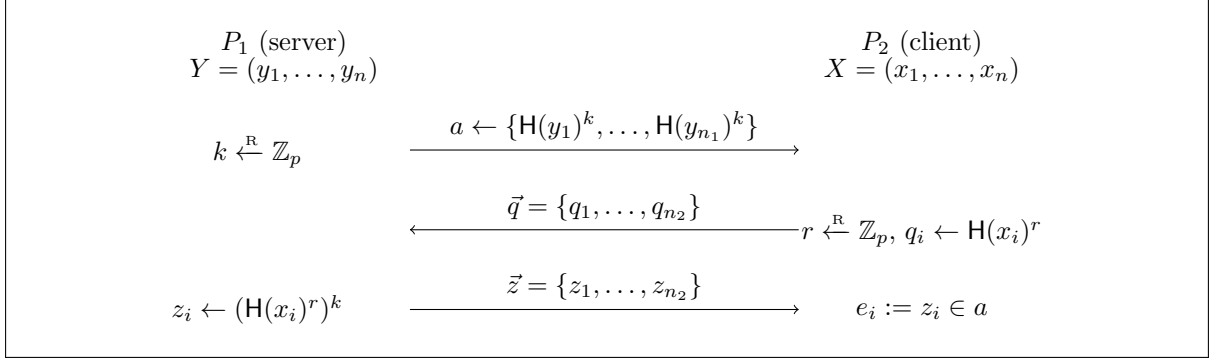


Figure 14: Sigma-mqPMT based on the DDH assumption

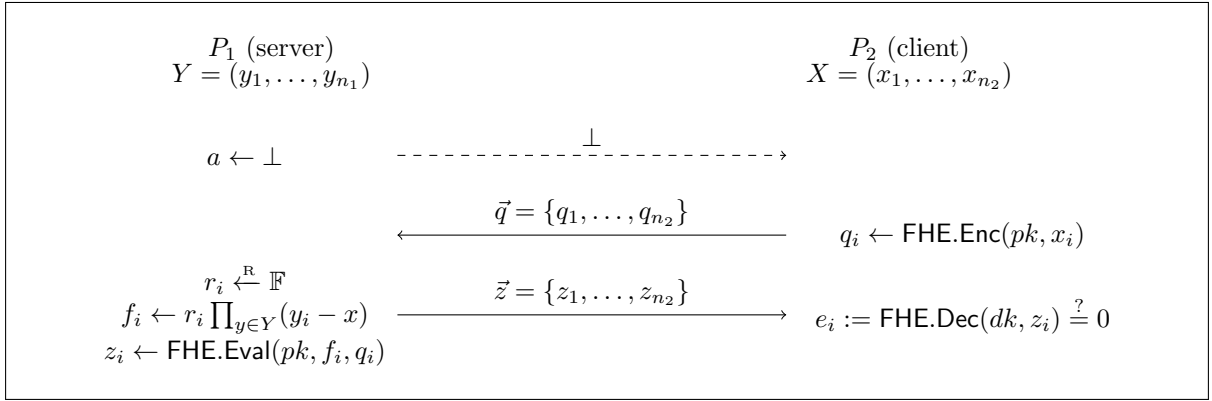


Figure 15: Sigma-mqPMT based on FHE

## 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].

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:  $\{\text{AHE.Enc}(pk, x_i^1), \dots, \text{AHE.Enc}(pk, x_i^{n_1})\}$ .

*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  $\text{Sim}_{P_1}$  and  $\text{Sim}_{P_2}$  for simulating corrupt  $P_1$  and  $P_2$  respectively, and argue the indistinguishability of produced transcript from the real execution.

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

Hybrid<sub>0</sub>:  $P_2$ 's view in the real protocol.

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

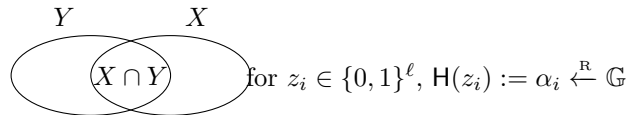
Clearly,  $\text{Sim}_{P_2}$ 's simulated view is identical to the real view.

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

Hybrid<sub>0</sub>:  $P_1$ 's view in the real protocol.

Hybrid<sub>1</sub>: Given  $P_1$ 's output  $k$  and  $\pi$ ,  $\text{Sim}_{P_1}$  chooses the randomness  $s$  for  $P_2$ , and simulates with the knowledge of  $X = (x_1, \dots, x_n)$ :

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



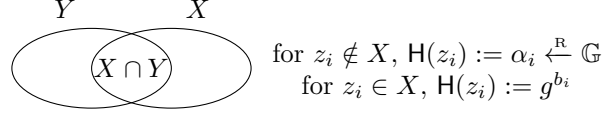
Clearly,  $\text{Sim}_{P_1}$ 's simulated view in Hybrid<sub>1</sub> is identical to  $P_1$ 's real view.

Hybrid<sub>2</sub>:  $\text{Sim}_{P_1}$  does not choose the randomness for  $P_2$ , and simulates without the knowledge of  $X$ . It honestly emulates random oracle  $H$  as in Hybrid<sub>1</sub>, and only changes the simulation of  $P_2$ 's message.

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

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

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



Clearly, if  $c_i = ab_i$ ,  $\mathcal{A}$  simulates  $\text{Hybrid}_1$ . Else, it simulates  $\text{Hybrid}_2$ . Thereby,  $\text{Sim}_{P_1}$ '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  $\text{Hybrid}_1$  and  $\text{Hybrid}_2$  to the pseudorandomness of  $F_k(\text{H}(\cdot))$ , which is in turn based on the DDH assumption.