Charge Your Clients: Payable Secure Computation and Its Applications

Cong Zhang[1,2], Liqiang Peng[3], Weiran Liu[3], Shuaishuai Li[4], Meng Hao[5(✉)], Lei Zhang[3], and Dongdai Lin[6,7]

[1] Institute for Advanced Study, BNRist, Tsinghua University, Beijing 100084, China
[2] State Key Laboratory of Cryptography and Digital Economy Security, Tsinghua University, Beijing, 100084, China
zhangcong@mail.tsinghua.edu.cn
[3] Alibaba Group
{plq270998,weiran.lwr}@alibaba-inc.com, zongchao.zl@taobao.com
[4] Zhongguancun Laboratory
liss@zgclab.edu.cn
[5] Singapore Management University
menghao303@gmail.com
[6] State Key Laboratory of Cyberspace Security Defense, Institute of Information Engineering, Chinese Academy of Sciences, China
[7] School of Cyber Security, University of Chinese Academy of Sciences, China
ddlin@iie.ac.cn

Abstract. The online realm has witnessed a surge in the buying and selling of data, prompting the emergence of dedicated data marketplaces. These platforms cater to servers (sellers), enabling them to set prices for access to their data, and clients (buyers), who can subsequently purchase these data, thereby streamlining and facilitating such transactions. However, the current data market is primarily confronted with the following issues. Firstly, they fail to protect client privacy, presupposing that clients submit their queries in plaintext. Secondly, these models are susceptible to being impacted by malicious client behavior, for example, enabling clients to potentially engage in arbitrage activities.

To address the aforementioned issues, we propose payable secure computation, a novel secure computation paradigm specifically designed for data pricing scenarios. It grants the server the ability to securely procure essential pricing information while protecting the privacy of client queries. Additionally, it fortifies the server's privacy against potential malicious client activities. As specific applications, we have devised customized payable protocols for two distinct secure computation scenarios: Keyword Private Information Retrieval (KPIR) and Private Set Intersection (PSI).

We implement our two payable protocols and compare them with the state-of-the-art related protocols that do not support pricing as a baseline. Since our payable protocols are more powerful in the data pricing setting, the experiment results show that they do not introduce much overhead over the baseline protocols. Our payable KPIR achieves the same online cost as baseline, while the setup is about $1.3 - 1.6\times$ slower than it. Our payable PSI needs about $2\times$ more communication cost than that of baseline protocol, while the runtime is $1.5 - 3.2\times$ slower than it depending on the network setting.

1 Introduction

1.1 Background

The storage and analysis of data has seen dramatic growth in recent years. Organizations have never valued data more highly. Many products and services are delivered purely in digital forms. Many big data applications are built on the second use or reuse of data [vdSDLP19], that is, the same data are customized and reused by many applications for different purposes. In economic activities where data are shared, exchanged and reused, it is essential to measure the value of data properly, and this has led to the appearance of data markets [CKK19, FSF20]. Subsequently, data pricing was widely studied [KUB+12a, KUB+12b, KUB+13, CKK17, DK17, CKK19, CYW+22].

However, current data pricing models[8] assume that the client (data buyer) submits its query q to the server (seller in the data market), who then prices the data based on the query q. This model directly reveals the client's private query q to the server. In practical scenarios, the clients may not want to disclose the information they query about. For example, a retail company wants to ask for the sales of a certain product on an E-commerce platform. But they want to keep the product they are querying about private because this information may include their future sales strategy. A helpful way is to use secure multi-party computation (MPC) protocols [Yao86, GMW87] to protect the query privacy of clients, so that the server cannot learn any information about the client's query after the query is completed. Consider the following use cases.

Literature purchase. Consider such a scenario: The server has a literature database, where each data item consists of a literature identifier (such as title or doi) and corresponding literature. The client wants to purchase the literature it needs from the server. The client has the identifier of the literature as the keyword and wants to use the keyword to retrieve the corresponding literature. The server does not want to disclose any information other than the client's required literature, while the client does not want to reveal the queried literature's identifier to the server. By using a (symmetric) keyword private information retrieval (KPIR) protocol, the client can retrieve the value associated with a query while the server knows nothing about the client's query.

Accurate advertising placement. Considering that the server is an advertiser with a set of users who have strong shopping premises and high purchasing power. The client is an enterprise that wants to advertise to users and has a set of users who want to advertise to them. The client does not want to advertise all users in its set, which leads to high costs. Therefore, the client wants to know the users who have a high willingness to pay, and wants to advertise to these specific users. By using a private set intersection (PSI) protocol, the client could obtain the intersection of its user set and the server's set, while the server knows nothing about the client's set.

By using the MPC protocol mentioned above, the client can obtain the desired data while the server learns nothing about the client's input/output. However, this approach also raises difficulties for charging. Since the server knows nothing about the client's output, a malicious client may lie to the server that it receives nothing to avoid payment. Fernandez [Fer22] showed that a strategic client might try to improve their benefits through malicious behavior. This is a blow to the server's enthusiasm in the data market. Consequently, we ask the following questions:

Can we introduce the concept of charging into secure computation? If possible, can we design concrete protocols that support charging/payment in secure computation?

1.2 Our Contribution

In this paper, we answer the above two questions affirmatively. Our contribution can be summarized as follows:

1. We propose a new secure computation definition in the data pricing scenario called Payable Secure Computation (PSC). It returns an additional *judge* information to the server after the server and the client execute the protocol, and the server can charge the client based on this information.
2. We present payable protocols for two concrete scenarios of secure computation: Keyword Private Information Retrieval (KPIR) and Private Set Intersection (PSI). For KPIR, our protocol allows the server to additionally obtain a bit, indicating whether the client's query is in the database. If so, the client will be charged. For PSI, our protocol allows the server additionally obtain the intersection size, and the server can charge the client based on this information. The larger the intersection size, the more the client needs to pay.
3. We implement our Payable KPIR and Payable PSI protocols and compare them with the state-of-the-art related protocols that do not support pricing as a baseline. Concretely, we compare our payable KPIR protocol with the KPIR protocol [CMdG+21], and the experiments show that our payable KPIR achieves the same online cost as [CMdG+21], while the setup is about $2\times$ slower than it. We compare our payable PSI protocol with the state-of-the-art PSI cardinality (PSI-CA) protocol [GMR+21], the experiments show that our payable PSI needs about $2\times$ more communication cost than that of PSI-CA protocol [GMR+21], while the runtime is $1.5 - 6.2\times$ slower than that of PSI-CA depending on the network setting.

[8] For example, the AWS Data Exchange platform [aws].

1.3 Technique Overview

We now provide the main intuition of our payable secure computation definition and the high-level technical overview for our payable KPIR and PSI constructions.

Payable Secure Computation. Let's consider a general scenario where a server has a database, and a client wants to obtain certain data from this database. They need to execute a protocol. After the protocol, the client may obtain the desired data or nothing (if the server does not have the data the client wants). Now, we need to define a new *payable* functionality to enable the server to charge the client and design a protocol to implement this new functionality. For server privacy, the protocol should satisfy security against the malicious client since the client may engage in malicious behavior during protocol execution to gain profits. For example, the client may buy more data at a lower price[9] or deceive the server that it does not obtain the data. For client privacy, we only need the protocol to satisfy security against the semi-honest server. Because we generally believe that the server is a large data provider, and the server cannot afford the embarrassment, loss of reputation, and negative press associated with being caught cheating. To enable the server to charge, we also need to provide the server with pricing information after the protocol, which we denote as *judge*. We note that *judge* can be computed from the output of the client, but can only contain the minimum information required for pricing and cannot disclose additional client output. For the correctness of the protocol, we need to ensure that the server's charges are reasonable, that is, the charges must meet a certain charging strategy agreed upon by both parties. When a client engages in malicious behavior, we only require that the client cannot benefit itself, but we do not fully require that the client fulfill the protocol. For example, we allow the client to engage in malicious behavior, but the only result is that the client will be overcharged.

Combining the above ideas together, we obtain the definition of payable secure computation. See Section 3 for details.

Payable KPIR. In KPIR, a client wants to retrieve the value associated with a certain key in the server's databases, which consist of key-value pairs. To make the server charge the client, we consider letting the server obtain information on whether the client retrieves a value, that is, $judge = 1$ if and only if the client's key is in the database. Note that the traditional KPIR that only protects the client's privacy cannot be used, as we require both the server and client privacy.

Our start point is the KPIR protocol of Freedman et al. [FIPR05] (under the name keyword search). They proposed a method of using Oblivious Pseudo-Random Function (OPRF) to upgrade any semi-private KPIR protocol into a fully-private one. The main idea is as follows, the server picks a random PRF key s and updates its database from $\{(x_i, v_i)\}_{i\in[n]}$ to $\{(x'_i, c_i)\}_{i\in[n]}$, where $F_s(x_i) = x'_i||x''_i, c_i = v_i \oplus x''_i$. Then, the parties invoke OPRF functionality with server input s and client input query q. As a result, the client obtains $F_s(q) = q'||q''$. Finally, the parties execute the semi-private KPIR protocol with server input $\{(x'_i, c_i)\}_{i\in[n]}$ and client input query q'. The client learns whether $q' \in \{x'_i\}_{i\in[n]}$, and if so, also learns the corresponding c. If $q' \in \{x'_i\}_{i\in[n]}$, the client outputs $v := c \oplus q''$, otherwise, it outputs $\perp$.

To let the server obtain the *judge*, we first try to interpret the above protocol at an abstract level. Our main observation is that the usage of OPRF is two-fold. Firstly, the first half x' of OPRF is used as the new keyword. Secondly, the second half x'' of OPRF is used to derive n pseudorandom one-time pads for encrypting values. The client can obtain the new keyword and corresponding value decryption key in OPRF at the same time.

Based on the above new interpretation, our idea is to split the OPRF into two sub-OPRFs and execute them separately. We let the server select *two* PRF keys s^k and s^v. The first key s^k is used to generate a new keyword $x' = F_{s^k}(x)$ and the second key s^v is used to derive one-time pad $x'' = F_{s^v}(x)$. The protocol is as follows, the parties invoke the first OPRF with server input s^k and client input query q. As a result, the client obtains $F_{s^k}(q) = q'$. Then, the server updates the database to $\{(x'_i, c_i)\}_{i\in[n]}$ as before, and the parties invoke a semi-private KPIR protocol with server input $\{(x'_i, c_i)\}_{i\in[n]}$ and client input query q'. Now, if the client learns $\perp$, that is, the query is not in the database, the client just sends a 0 to the server, indicating that the query is not in the database. Otherwise, if the client obtains an encrypted value c, the client uses the input q to invoke the second OPRF with the server, and the server inputs the second PRF key s^v. As a result, the client obtains $q'' = F_{s^v}(q)$ and outputs

[9] This behaviour is called arbitrage in data pricing models [FV12, LK14].

$v := q'' \oplus c$. In this way, the server sets $judge = 0$ if it receives 0 after the KPIR and $judge = 1$ if it executes a second OPRF. The high-level idea of our payable KPIR is shown in Figure 1.

Note that the above protocol may still be vulnerable to malicious client attacks. For example, even if a malicious client receives $\perp$ in the KPIR of the second step, it can still execute the second OPRF with the server. However, this type of attack does not bring any benefits to the client. On the contrary, it only makes the server believe that the client has obtained the data and charges the client. In fact, the client has yet to get any valuable data. The only method for the client to obtain the desired data is to execute the second OPRF with the server, which will inevitably cause the server to set $judge = 1$. Consequently, the above protocol already meets the requirement of payable property. We give our formal description of our payable KPIR and the detailed security analysis in Section 4.

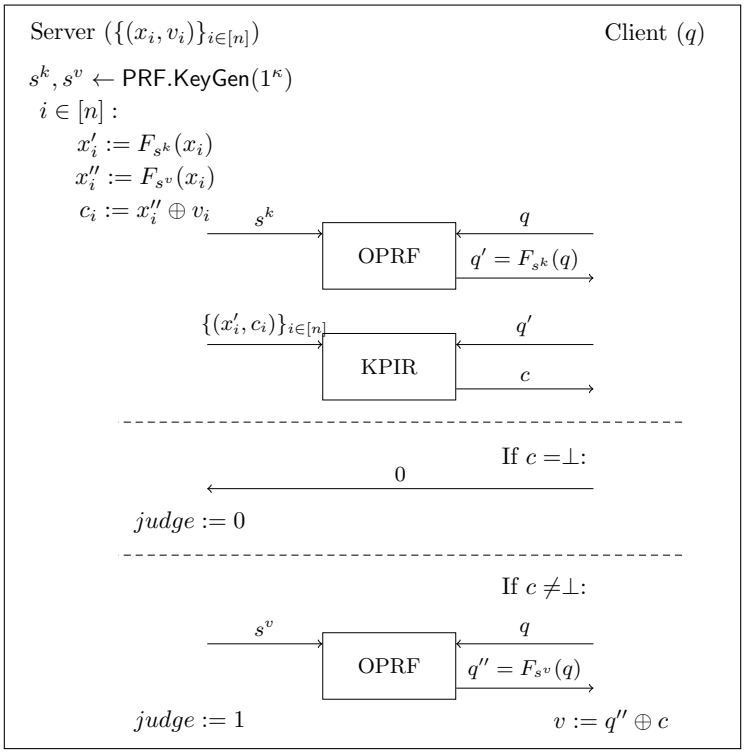

Fig. 1: Core idea of our payable KPIR protocol

Payable PSI. In PSI, a client with a set Y wants to obtains the intersection $Y \cap X$ with server's set X. For the server to charge the client, we let the server obtain the intersection size, that is, $judge = |X \cap Y|$. We note that it is reasonable to charge based on the size of the intersection: the larger the intersection, the more data the client receives, hence the server charges a higher fee.

We investigated the state-of-the-art PSI protocols [KKRT16, CM20, RS21, RT21, RR22] and found that they are all based on multi-query Private Membership Test (mq-PMT). In mq-PMT, a server inputs a set X, a client inputs a set of element $\{y_i\}_{i\in[n]}$ and learns whether $y_i \in X$ for $i \in [n]$. As a result, the client learns intersection directly from the output of mq-PMT. We note that mq-PMT can be easily obtained by OPRF: the server inputs a PRF key k while the client obtains $\{F_k(y_i)\}_{i\in[n]}$ on its input $\{y_i\}_{i\in[n]}$, then the server computes and sends $\{F_k(x)\}_{x\in X}$ to the client. The client tests whether $F_k(y_i) \in \{F_k(x)\}_{x\in X}$ to determine whether $y_i \in X$ for $i \in [n]$.

To construct payable PSI, the first question is how to make the server obtain the size of the intersection, i.e., intersection cardinality. We investigated techniques used in other private set operation protocols. We found that there are two main ideas to transform PSI/mq-PMT to the protocol only outputs intersection cardinality to one party. The first is called *permuted* mq-PMT. It outputs the

intersection cardinality to the client[10]. The main idea is to let the server additionally input a random permutation π, and the client learns whether $y_{\pi(i)} \in X$ for $i \in [n]$. Note that the client could only obtain intersection cardinality because it knows nothing about the permutation π. The second is called multi-query *Reverse* Private Membership Test (mq-RPMT). It outputs the intersection cardinality to the server. The main idea is to let the server instead of the client learn whether $y_i \in X$. In this way, the server obtains intersection cardinality because it has no information about Y. Chen et al. [CZZ+24] provided a detailed classification for the family of PMT protocols and studied their relationships in the semi-honest setting.

Although permuted mq-PMT/mq-RPMT somehow satisfies our requirement, there are still two challenges. Firstly, it has not fully completed the task, and we still need to consider how to enable the client to obtain the intersection. Secondly, all known efficient permuted mq-PMT/mq-RPMT constructions only satisfy semi-honest security, while we need our protocol to satisfy security against malicious client.

Note that solving the first challenge is trivial in mq-RPMT: we can let the server send the indication bits to the client after mq-RPMT. Since the server is honest, the client can directly obtain the intersection from these bits. Unfortunately, we investigated the existing mq-RPMT protocols and found that no protocol meets the security against malicious client, and it doesn't seem to be fixable by a small tweak of existing mq-RPMT. We have listed the existing mq-RPMT protocols and provided a detailed explanation of why they do not meet the security against malicious clients in Appendix A.

As a result, we investigated the existing permuted mq-PMT protocols and found that there are two candidate permuted mq-PMT protocols that could be transformed into payable PSI. The first is the permuted mq-PMT protocol proposed by Jia et al. [JSZ+22]. We give a brief review of their permuted mq-PMT as follows. Their protocol employs an Oblivious Switching Network (OSN) [MS13] functionality and an OPRF functionality. In OSN, a sender inputs a set $X = \{x_i\}_{i\in[n]}$, and a receiver inputs a permutation π over $[n]$. As a result, the two parties obtain the secret sharing of $\pi(X)$, namely, the sender obtains $\{a_i\}_{i\in[n]}$ and the receiver obtains $\{a'_i\}_{i\in[n]}$, satisfying $a_i \oplus a'_i = x_{\pi(i)}$. Their permuted mq-PMT protocol executes as follows with server $\mathcal{S}$'s input X and client $\mathcal{C}$'s input Y: $\mathcal{S}$ and $\mathcal{C}$ execute the OSN protocol first, where the server $\mathcal{S}$ inputs X and receives $\{a_i\}_{i\in[n]}$, the client $\mathcal{C}$ inputs a random permutation π and obtains $\{a'_i\}_{i\in[n]}$. Next, $\mathcal{S}$ and $\mathcal{C}$ execute the OPRF protocol. The client $\mathcal{C}$ picks a random PRF key k as input. The server $\mathcal{S}$ inputs $\{a_i\}_{i\in[n]}$, and receives $\{F_k(a_i)\}_{i\in[n]}$. Now, $\mathcal{C}$ defines sets $P_i := \{F_k(a'_i \oplus y)\}_{y\in Y}$ and sends $\{P_i\}_{i\in[n]}$ to the server. After receiving $\{P_i\}_{i\in[n]}$, $\mathcal{S}$ defines $U[i] = 1$ if $F_k(a_i) \in P_i$ and $U[i] = 0$ otherwise. If $x_{\pi(i)} \in Y$, there exists $y_j \in Y$ such that $x_{\pi(i)} = y_j$, we have $F_k(a_i) = F_k(x_{\pi(i)} \oplus a'_i) = F_k(y_j \oplus a'_i) \in P_i$. If $x_{\pi(i)} \notin Y$, the pseudorandomness of PRF guarantees that the probability of $F_k(a_i) \in P_i$ is negligible.

To let the client obtain the intersection, we observe that the parties already have the additive sharing of $\{x_{\pi(i)}\}$. We simply let the server send the a_i corresponding to $U[i] = 1$ to the client. The client could recover the intersection from $x_{\pi(i)} = a_i \oplus a'_i$. To meet the security against malicious client, we should replace the underlying OPRF and OSN protocols with the malicious secure ones. However, all the OSN protocols known to date only consider semi-honest security. Fortunately, we prove that the semi-honest OSN protocol [MS13] is still secure against the malicious receiver. The high-level idea of this payable PSI is shown in Figure 2.

Note that the above protocol requires $O(n^2)$ communication since each P_i contains n values. Jia et al. [JSZ+22] showed how to improve the communication by cuckoo hashing technique [PR04]. However, this optimization is problematic when recovering the intersection. The client may learn the cuckoo hash positions of the server's items, which may reveal information about the server's entire input. To address this problem, we consider having the parties apply an OPRF to their input items before permuted mq-PMT. As a result, the position information now is only related to this pseudorandom set, which can be simulated. The same technique was also used in the construction of unbalanced PSI by Chen et al. [CHLR18].

The second candidate of permuted mq-PMT is proposed by Cristofaro et al. [CGT12]. The main idea dates back to the DH-PSI [Mea86, HFH99]. To satisfy security against the malicious client, we adapt the zero knowledge proof techniques of [MPR+20] to this protocol. We note that this protocol is inefficient in practice due to the use of a large amount of algebraic zero knowledge proofs, and we only

[10] In fact, the client in permuted mq-PMT plays the role of server in our payable PSI, since we require the server to obtain the intersection cardinality.

consider it as a theoretical construction. The advantage of this protocol is that the communication is linear $O(n)$. And it can be easily transformed into a protocol that is secure against malicious servers by letting the server also send the zero knowledge proofs. We give our formal descriptions of our two payable PSI and the detailed security analysis in Section 5.

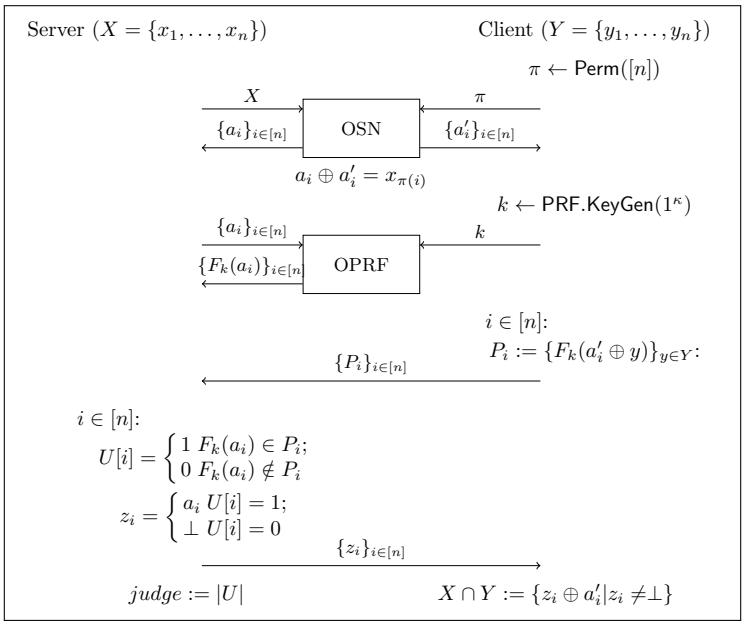

Fig. 2: Core idea of our payable PSI protocol

1.4 Related Work

Data Pricing. To understand what are sold in data markets and for what purposes, Muschalle et al. [MAP12] consider the common queries and demands on data markets, as well as the pricing strategies and models. Versioning is an important strategy in data pricing. The server can customize data into different versions according to clients' needs. One way to form multiple versions of data products is query-based pricing [KUB+12a, KUB+12b, KUB+13, KUB+15]. Koutris et al. [KUB+12a, KUB+15] propose a framework of query and view based data pricing. The major idea is that the server only needs to specify the prices on a few views, and then the prices of other views can be decided algorithmically. They also developed an integer linear programming formulation for the pricing problem with a large number of queries. With the application of machine learning (ML) in the field of data analysis, Chen et al. [CKK17, CKK19] proposed model-based pricing. Their key observation is that ML users typically need only as much data as needed to meet their accuracy goals, which leads to new tradeoffs between price, accuracy, and runtimes.

Keyword Private Information Retrieval. A trivial method to construct KPIR is to let the client store a mapping of all key-value pairs. Based on this mapping, the client finds the index corresponding to its query and then retrieves the desired value by calling a standard index PIR scheme. Unfortunately, this requires the client to store mappings that are linear in the database size. Chor et al. [CGN98] considered to let the user interactively query the server to obtain the physical address of the desired value privately. With the physical address, the user then conducts index PIR to retrieve the desired value. Ali et al. [ALP+21] considered using Cuckoo hashing [PR04] to map each data value to a hashing table, the client then retrieves each location of hashing table corresponding to its query using index PIR. Mahdavi and Kerschbaum [MK22] using constant-weight code to encode the key, and let the server compute the equality operators over the ciphertext of the client's encoding. Ahmad et al. [AAAG22] followed the constant-weight code paradigm and further optimized the computation overhead in equality operators. Patel et al. [PSY23] proposed a KPIR scheme that invokes underlining

FHE-based index PIR scheme in a non-black-box way. Their scheme achieves almost the same overhead as the state-of-the-art index PIR scheme.

Private Set Intersection. The original PSI protocol [Mea86, HFH99] is based on Diffie-Hellman key exchange, which is denoted as DH-PSI. Subsequently, [CKT10, JL10, RT21] adapted DH-PSI to malicious security. DH-PSI is considered one of the PSI protocols with the lowest communication overhead. However, its computation efficiency is low in practice due to the need to perform public key operations on each element.

Freedman et al. [FNP04] proposed an oblivious polynomial evaluation (OPE) based PSI protocol. The main idea is to use polynomials to represent sets, that is, a polynomial whose roots are set elements. The intersection can be obtained from the oblivious evaluation of the server's polynomial under the client's elements. This technique was later extended to the malicious setting [DMRY09, FHNP16, CILO22]. Protocol constructions along this roadmap often use additive homomorphic encryption (AHE), which is inefficient in practical.

The current state-of-the-art PSI protocols [KKRT16, PRTY19, CM20, RS21, GPR+21, RR22] are based on Oblivious Transfer (OT) extension [IKNP03], which generates a large number of OTs using a small number of base OTs in combination with symmetric operations. The main idea behind this paradigm is to adapt OT extension protocols to OPRF protocol, and transform OPRF to the mq-PMT, as we mentioned before. However, the main drawback of these techniques is that they are less modular and thus more difficult to modify to accommodate other PSI variants, e.g., PSI cardinality (PSI-CA).

2 Preliminaries

2.1 Notation

We use κ and λ to denote the computational and statistical security parameters, respectively. We use $[n]$ to denote the set $\{1, 2, \ldots, n\}$. For a bit string v we let v_i denote the ith bit. We use the abbreviation PPT to denote probabilistic polynomial-time. We say that a function f is negligible in κ if it vanishes faster than the inverse of any polynomial in κ, and write it as $f(\kappa) = \mathsf{negl}(\kappa)$. By $a \xleftarrow{R} A$, we denote that a is randomly selected from the set A, $a \leftarrow \mathsf{A}(x)$ denotes that a is the output of the randomized algorithm A on input x, and $a := b$ denotes that a is assigned by b.

2.2 Security Model

We use the standard security definition for two-party computation [Gol04, Lin17] in this work. Since our protocol requires security against semi-honest server and malicious client, we give definitions of semi-honest security and malicious security as follows.

Let $\mathcal{F}$ be a functionality between a server $\mathcal{S}$ with input $I_{\mathcal{S}}$ and a client $\mathcal{C}$ with input $I_{\mathcal{C}}$. Let Π be a two-party protocol for computing $\mathcal{F}$.

Semi-honest Security. Let $\mathsf{view}_{\mathcal{P}}^{\Pi}(I_{\mathcal{S}}, I_{\mathcal{C}})$ be the views of party $\mathcal{P}$ ($\mathcal{P} \in \{\mathcal{S}, \mathcal{C}\}$) in the protocol, which consists of $\mathcal{P}$'s input, randomness tape, and received messages during the protocol. Let $\mathsf{output}(I_{\mathcal{S}}, I_{\mathcal{C}})$ be the output of both parties in the protocol.

Definition 1. *A protocol Π is said to securely compute functionality $\mathcal{F}$ against semi-honest $\mathcal{P}$ if for every PPT adversary $\mathcal{A}$ that corrupt $\mathcal{P}$, there exists a PPT simulator $\mathsf{Sim}_{\mathcal{P}}$ such that for all inputs $I_{\mathcal{S}}$ and $I_{\mathcal{C}}$,*

$$\{\mathsf{view}_{\mathcal{P}}^{\Pi}(I_{\mathcal{S}}, I_{\mathcal{C}}), \mathsf{output}(I_{\mathcal{S}}, I_{\mathcal{C}})\} \approx_c \{\mathsf{Sim}_{\mathcal{P}}(I_{\mathcal{P}}, \mathcal{F}(I_{\mathcal{S}}, I_{\mathcal{C}})), \mathcal{F}(I_{\mathcal{S}}, I_{\mathcal{C}})\}$$

Malicious Security. We use the real-ideal paradigm to define malicious security. In the ideal execution, the parties privately send their inputs to the ideal functionality. The functionality simply computes $\mathcal{F}(I_{\mathcal{S}}, I_{\mathcal{C}})$ and returns the result to the parties. Let Sim be a PPT adversary corrupt party $\mathcal{P}$ in the ideal world, we use $\mathsf{Ideal}_{\mathcal{F},\mathsf{Sim},\mathcal{P}}(I_{\mathcal{S}}, I_{\mathcal{C}})$ to denote the output pair of the honest party and the adversary Sim from the above ideal execution. In the real execution, there is no trusted party. Instead, the parties communicate with each other using a protocol Π. Let $\mathcal{A}$ be a PPT adversary corrupt party $\mathcal{P}$ in the real world, we use $\mathsf{Real}_{\Pi,\mathcal{A},\mathcal{P}}(I_{\mathcal{S}}, I_{\mathcal{C}})$ to denote the output pair of the honest party and the adversary $\mathcal{A}$ from the above real execution.

Definition 2. *A protocol Π is said to securely compute functionality $\mathcal{F}$ against malicious $\mathcal{P}$ if for every PPT adversary $\mathcal{A}$ that corrupt $\mathcal{P}$ in real world, there exists a PPT simulator $\mathsf{Sim}_{\mathcal{P}}$ such that for all inputs $I_{\mathcal{S}}$ and $I_{\mathcal{C}}$,*

$$\{\mathsf{Ideal}_{\mathcal{F},\mathsf{Sim},\mathcal{P}}(I_{\mathcal{S}}, I_{\mathcal{C}})\} \approx_c \{\mathsf{Real}_{\Pi,\mathcal{A},\mathcal{P}}(I_{\mathcal{S}}, I_{\mathcal{C}})\}$$

2.3 Oblivious PRF

An oblivious PRF (OPRF) [FIPR05] is a 2-party protocol between a sender and a receiver, where the sender inputs a PRF key k and receives nothing, while the receiver inputs a set of queries $\{x_i\}_{i\in[n]}$ and obtains PRF values $\{F_k(x_i)\}_{i\in[n]}$.

The ideal functionality for OPRF is shown in Figure 3.

Parameters: Sender $\mathcal{S}$, Receiver $\mathcal{R}$, a PRF F.
Functionality:

- Wait for input $\{x_1, \ldots, x_n\}$ from the receiver $\mathcal{R}$.
- Wait for input k from the sender $\mathcal{S}$.
- Give $\{F_k(x_1), \ldots, F_k(x_n)\}$ to the receiver $\mathcal{R}$.

Fig. 3: Oblivious PRF Functionality $\mathcal{F}_{\mathsf{oprf}}$

2.4 Oblivious Switching Network

An oblivious switching network (OSN) [MS13] works as follows. The sender inputs a set of elements $X = \{x_i\}_{i\in[n]}$ and the receiver inputs a permutation π over $[n]$. As a result, the parties obtain the additive shares of $\pi(X) = \{x_{\pi(i)}\}_{i\in[n]}$, that is, the sender receives $\{a_i\}_{i\in[n]}$ and the receiver receives $\{b_i\}_{i\in[n]}$, where $a_i \oplus b_i = x_{\pi(i)}$ for $i \in [n]$. The formal definition of OSN functionality is given in Figure 4.

The original OSN protocol [MS13] works by considering a universal switching network (i.e., Waksman or Beneš network), which consists of $O(n \log n)$ 2-input, 2-output switches. They use oblivious transfer (OT) to achieve secret sharing for each switching gate. We note that their protocol only achieves semi-honest security, while we need the security against *malicious* receiver. Fortunately, we find that the original OSN protocol is naturally secure against malicious receiver if we use the malicious secure OT as its building block, which exactly meets our requirement. We give a detailed description of the OSN protocol [MS13] and the security proof against malicious receiver in Appendix B.

Parameters: Sender $\mathcal{S}$, Receiver $\mathcal{R}$, set size n.
Functionality:

- Wait for input π from the receiver $\mathcal{R}$.
- Wait for input $X = \{x_1, \ldots, x_n\} \subset \{0,1\}^l$ from the sender $\mathcal{S}$.
- Pick random $a_i \leftarrow \{0,1\}^l$ and compute $b_i = a_i \oplus x_{\pi(i)}$ for $i \in [n]$.
- Give $\{a_i\}_{i\in[n]}$ to the sender $\mathcal{S}$ and give $\{b_i\}_{i\in[n]}$ to the receiver $\mathcal{R}$.

Fig. 4: Oblivious switching network Functionality $\mathcal{F}_{\mathsf{osn}}$

2.5 Private Set Intersection

Private Set Intersection (PSI) [FNP04] is a secure computation protocol that allows two parties, the server and the client, to compute the intersection of their private sets X and Y, such that the client only learns $X \cap Y$ from the interaction and the server learns nothing. The ideal functionality for PSI is given in Figure 5.

Parameters: Server $\mathcal{S}$, Client $\mathcal{C}$, set sizes n.
Functionality:

- Wait for input $X = \{x_1, \ldots, x_n\}$ from the server $\mathcal{S}$.
- Wait for input $Y = \{y_1, \ldots, y_n\}$ from the client $\mathcal{C}$.
- Give output $X \cap Y$ to the client $\mathcal{C}$.

Fig. 5: Private Set Intersection Functionality $\mathcal{F}_{\mathsf{psi}}$

2.6 Keyword Private Information Retrieval

Private Information Retrieval (PIR) [CKGS98] enables a client, holding an index i, to retrieve the i-th entries from a database holding by a server, while hiding the index i from the server. In most practical applications, databases more closely resemble key-value pairs where users want to retrieve the value associated to a certain key. Therefore, the variant Keyword Private Information Retrieval (KPIR) [CGN98] considers that the client query consists of a keyword instead of the index of the entry in the database.

Though PIR is in fact a two-party protocol, almost all known PIR schemes [MBFK16, ACLS18, ALP+21, MCR21, MK22, MW22, HHC+23, PSY23, DPC23] are defined as three algorithms, namely $(\mathsf{Query}, \mathsf{Answer}, \mathsf{Recover})$. The behavior of the protocol is fixed to the following two rounds of interaction: the client first generates a query message through the Query algorithm and sends it to the server. After receiving the query message, the server generates a response through the Answer algorithm and sends the response to the client. Then the client uses the $\mathsf{Recover}$ algorithm to obtain the output. The security of PIR only considers client privacy, which requires that the query messages generated by distinct queries are indistinguishable. A variant of PIR called symmetric PIR [GIKM00] also considers server privacy, that is, the client knows nothing about the database except that the data it retrieves.

However, the above definition is too restrictive for the design of PIR, and its security definition only corresponds to the semi-honest security in the simulation-based security definition, that is, the adversaries may try to learn as much information as possible from a given protocol execution but are not able to deviate from the protocol steps. This is in contrast to malicious adversaries which are able to deviate arbitrarily from the protocol. To broaden our design and achieve higher security, we define the ideal functionality of KPIR and design a protocol to implement this functionality, instead of three algorithms. We note that this functionality naturally captures the privacy of both the server and the client.

The ideal functionality for KPIR is given in Figure 6.

Parameters: Server $\mathcal{S}$, Client $\mathcal{C}$, database sizes n.
Functionality:

- Wait for input $(X, V) = \{(x_1, v_1), \ldots, (x_n, v_n)\} \subset \{0,1\}^* \times \{0,1\}^l$ from the server $\mathcal{S}$.
- Wait for input $q \in \{0,1\}^*$ from the client $\mathcal{C}$.
- Give output v_i to the client $\mathcal{C}$ such that $q = x_i$, or $\perp$ if no such i exists.

Fig. 6: Keyword Private Information Retrieval Functionality $\mathcal{F}_{\mathsf{kpir}}$

3 Payable Secure Computation

To formally define payable secure computation, we first introduce the concept of *one-sided output functionality*, which can be viewed as the basic functionality for which we want to charge.

Definition 3 (One-sided output functionality). *Let f be a two-party functionality, if only one party receives output, we call the functionality f the one-sided output functionality. The party that receives output is called the client, and the other party is called the server.*

To enable the server to charge, we consider allowing the server to learn additional pricing information *judge*, which can be computed from the client's output. We define the payable functionality f' of one-sided output functionality f as follows.

Definition 4 (Payable functionality). *Let f be a one-sided output functionality. We call f' the payable functionality of f if it additionally delivers a judge, which could be computed from the client's output, to the server. If the client is corrupt, the functionality outputs a judge$'$ to the server according to the adversary's behavior.*

Now, we are ready to define payable secure computation. As discussed in Section 1.3, the protocol should securely compute f' against malicious client and semi-honest server. For correctness, we define a Check algorithm that can detect whether the pricing information *judge* is consistent with the output of the client when both parties behave honestly. We also define a Pricing algorithm that determines the final pricing. Note that we allow malicious clients to make an honest server learn an incorrect *judge*$'$, but we require that the pricing obtained based on *judge*$'$ must not be less than the pricing obtained based on *judge*, where *judge* is the output of the server when the client behaves honestly. That is, the behavior of a malicious client will only cause it to pay more. We give the formal definition of Payable Secure Computation (PSC) as follows.

Definition 5 (PSC). *A protocol π along with a pair of algorithms* (Check, Pricing) *is said to be a payable implement of a one-sided output functionality f if the following hold:*

- *The protocol π securely realizes the payable functionality f' of f against a malicious client.*
- *The protocol π securely realizes the payable functionality f' of f against a semi-honest server.*
- *If the honest server outputs judge and the honest client outputs m_c in an execution of the protocol, then we have* $\mathsf{Check}(judge, m_c) = 1$*, except with negligible probability.*
- *If the client is corrupt, let judge$'$ be the output of the server in this setting, then we have* $\mathsf{Pricing}(judge') \geq \mathsf{Pricing}(judge)$*, where judge is the server's output obtained from the interaction with an honest client.*

4 Payable KPIR

In this section, we propose payable KPIR and give the formal definition of payable KPIR functionality in Figure 7. As we discussed before, we want the server to charge the client when the client gets a value from the database. Therefore, we define $judge = 1$ if and only if the client's query is in the database. Note that this functionality allows the client to learn if its query is in the database. For malicious clients, we allow it to specify the *judge*. What we guarantee is that the client can only obtain the final value if and only if the client's query is in the database and it specifies the $judge = 1$. In other words, as long as the client receives a valid value, it must pay for it.

Parameters: Server $\mathcal{S}$, Client $\mathcal{C}$, database sizes n.
Functionality:

- Wait for input $(X, V) = \{(x_1, v_1), \ldots, (x_n, v_n)\} \subset \{0,1\}^* \times \{0,1\}^l$ from the server $\mathcal{S}$.
- Wait for input $q \in \{0,1\}^*$ from the client $\mathcal{C}$.
- Define $a = 1$ if and only if there exists x_i such that $q = x_i$ and return a to the client.
- Then:
  - If the client is honest, define $v = v_i$ if there exists x_i such that $q = x_i$ and $v = \perp$ otherwise. Define $judge = 1$ if $v \neq \perp$ and $judge = 0$ otherwise.
  - If the client is corrupt, receive a bit b from the adversary $\mathcal{A}$ and defines $judge = b$. If $b = 1$ and $a = 1$, waits for another bit b' from $\mathcal{A}$. If $b' = 1$ define $v = v_i$. Otherwise, define $v = \perp$.
- Give output v to the client $\mathcal{C}$ and give output *judge* to the server $\mathcal{S}$.

Fig. 7: Payable Keyword Private Information Retrieval Functionality $\mathcal{F}_{\text{p-kpir}}$

Now, we give our construction of payable KPIR protocol in Figure 8. For the Check algorithm, it output 1 if and only if v is consistent with *judge*, that is, it outputs 1 if and only if $v = \perp$ (resp.

$v \neq \perp$) and $judge = 0$ (resp. $judge = 1$). For the Pricing algorithm, we simply have the Pricing output the $judge$. Note that in payable KPIR, the fee is one-time. The pricing rules are determined by both parties in advance, and the server only needs to charge based on whether $judge = 1$.

Parameters:

- Two parties: server $\mathcal{S}$ and client $\mathcal{C}$.
- Ideal $\mathcal{F}_{\mathsf{oprf}}$ and $\mathcal{F}_{\mathsf{kpir}}$ primitives specified in Figure 3, and Figure 6, respectively.

Input of $\mathcal{S}$: $(X, V) = \{(x_1, v_1), \ldots, (x_n, v_n)\} \subset \{0,1\}^* \times \{0,1\}^l$
Input of $\mathcal{C}$: $q \in \{0,1\}^*$
Protocol:
Offline Stage:

1. The server $\mathcal{S}$ generates two PRF keys $s^k, s^v \leftarrow \mathsf{KeyGen}(1^\kappa)$ and computes $x'_i := F_{s^k}(x_i), x''_i := F_{s^v}(x_i)$ for $i \in [n]$. Then, $\mathcal{S}$ computes $c_i := x''_i \oplus v_i$ for $i \in [n]$.

Online Stage:

2. $\mathcal{S}$ and $\mathcal{C}$ invoke the OPRF functionality $\mathcal{F}_{\mathsf{oprf}}$. The server $\mathcal{S}$ acts as the sender in OPRF with input s^k. The client $\mathcal{C}$ acts as the receiver in OPRF with input q and receives $q' := F_{s^k}(q)$.
3. $\mathcal{S}$ and $\mathcal{C}$ invoke the KPIR functionality $\mathcal{F}_{\mathsf{kpir}}$. The server $\mathcal{S}$ acts as the server in KPIR with input $\{(x'_i, c_i)\}_{i\in[n]}$ and receives nothing. The client $\mathcal{C}$ acts as the client in KPIR with input q' and receives $\bar{c}$.
4. If $\bar{c} = \perp$, the client $\mathcal{C}$ sends 0 to the server and outputs $\perp$. The server $\mathcal{S}$ outputs $judge := 0$.
5. If $\bar{c} \neq \perp$, $\mathcal{S}$ and $\mathcal{C}$ invoke the OPRF functionality $\mathcal{F}_{\mathsf{oprf}}$. The server $\mathcal{S}$ acts as the sender in OPRF with input s^v and receives nothing. The client $\mathcal{C}$ acts as the receiver in OPRF with input q and receives $q'' := F_{s^v}(q)$. The client computes and outputs $v := q'' \oplus \bar{c}$. The server $\mathcal{S}$ outputs $judge := 1$.

Fig. 8: Payable Keyword Private Information Retrieval Protocol $\Pi_{\mathsf{p\text{-}kpir}}$

We give the correctness and the security analysis of our payable KPIR protocol as follows.

Correctness. If there exists x_i such that $q = x_i$, we have $F_{s^k}(q) = F_{s^k}(x_i)$ and $F_{s^v}(q) = F_{s^v}(x_i)$. Following the correctness of the underlining KPIR, the client obtains the value $c_i = F_{s^v}(x_i) \oplus v_i$ in step 3. Then, the client outputs $v_i := c_i \oplus F_{s^v}(q)$. Since the parties execute the second OPRF, the server sets $judge := 1$. If no such x_i exists, the correctness will only be violated when $\exists i \in [n]$, s.t. $F_{s^k}(x_i) = F_{s^k}(q)$. By setting the output length of $F_{s^k}(\cdot)$ to $\lambda + \log n$, a union bound shows that the probability of this event is negligible $2^{-\lambda}$. As the second OPRF is deemed unnecessary, the server set $judge := 0$.

Theorem 1. *The protocol in Figure 8 along with the above* (Check, Pricing) *algorithm is a payable implement of* $\mathcal{F}_{\mathsf{kpir}}$ *in the* $(\mathcal{F}_{\mathsf{oprf}}, \mathcal{F}_{\mathsf{kpir}})$*-hybrid model.*

Proof. For the corrupt *semi-honest* server, we exhibit the simulator $\mathsf{Sim}_{\mathcal{S}}(X, V, judge)$ as follows.

1. In the offline stage, the simulator $\mathsf{Sim}_{\mathcal{S}}$ generates $s^k, s^v, \{x'_i, x''_i, c_i\}_{i\in[n]}$ honestly.
2. In step 2, the simulator invokes the simulator of OPRF with input s^k and appends the output to the view.
3. In step 3, the simulator invokes the simulator of KPIR with input $\{(x'_i, c_i)\}_{i\in[n]}$ and appends the output to the view.
4. If $judge = 0$, the simulator appends 0 to the view. If $judge = 1$, the simulator invokes the simulator of OPRF with input s^v and appends the output to the view.

Now we argue that the view output by $\mathsf{Sim}_{\mathcal{S}}$ is indistinguishable from the real one. This is obtained by the underlying simulators' indistinguishability directly.

For the corrupt *malicious* client, we exhibit the simulator $\mathsf{Sim}_{\mathcal{R}}$ as follows:

1. In step 2, the simulator receives a query q from the adversary and delivers q to the ideal functionality. As a result, the simulator receives a bit a to indicate whether q is in the database. Then, the simulator selects a random q' and gives q' to the adversary.

2. In step 3, the simulator receives a query $\bar{q}'$. If $\bar{q}' = q'$ and $a = 1$, the simulator picks a random string $\bar{c}$ and sends it to the adversary. Otherwise, the simulator sends $\perp$ to the adversary.
3. Then, if the simulator receives 0 from the adversary, then it set $b = 0$ and sends it to the functionality. As a result, the simulator obtains $v = \perp$. If the adversary invokes OPRF functionality with input $\tilde{q}$, the simulator sets $b = 1$, and sends b to the functionality. Then, the simulator defines $b' = 1$ if $\tilde{q} = q$ and $b' = 0$ otherwise. The simulator sends b' to the functionality and obtains output v. If $b' = 1$ and $a = 1$, the simulator sends $v \oplus \bar{c}$ to the adversary. Otherwise, the simulator sends a random q'' to the adversary.

The correctness of this simulator directly follows the security of underlying protocols and the pseudorandomness of the PRF.

5 Payable PSI

In this section, we propose payable PSI and give the formal definition of payable PSI functionality in Figure 9. As we discussed before, we want the server to charge the client according to the cardinality of the intersection. Therefore, we define $judge = |X \cap Y|$.

Our payable PSI protocol is based on the permuted mq-PMT of Jia et al. [JSZ+22]. Since our protocol requires security against the malicious client, we prove the OSN protocol [MS13] (which only achieves semi-honest security) is also secure against malicious receiver[11]. To let the client obtain the intersection, we let the server send the permuted additive shares corresponding to the intersection elements to the client. We also use an OPRF to preprocess the sets of both parties to prevent the leak of cuckoo hashing positions of the intersection elements.

We also propose another payable PSI protocol based on the mq-PMT of Cristofaro et al. [CGT12]. The main idea dates back to the DH-PSI [Mea86, HFH99]. This protocol is inefficient in practice due to the use of a large amount of algebraic zero knowledge proofs, and we only consider it as a theoretical construction. The advantage of this protocol is that the communication is linear $O(n)$. And it can be easily transformed into a protocol that is secure against malicious servers by letting the server also send the zero knowledge proofs. We refer to Appendix C for more details.

Parameters: Server $\mathcal{S}$, Client $\mathcal{C}$, set sizes n.
Functionality:

- Wait for input $X = \{x_1, \ldots, x_n\}$ from the server $\mathcal{S}$.
- Wait for input $Y = \{y_1, \ldots, y_n\}$ from the client $\mathcal{C}$.
- Then:
  - If the client is honest, define $I = X \cap Y$ and $judge = |I|$.
  - If the client is corrupt, abort if receive abort from $\mathcal{A}$. Otherwise, define $I = X \cap Y$ and $judge = |I|$.
- Give output I to the client $\mathcal{C}$ and give output $judge$ to the server $\mathcal{S}$.

Fig. 9: Payable Private Set Intersection Functionality $\mathcal{F}_{\text{p-psi}}$

We give our construction of payable PSI protocol in Figure 10. For the Check algorithm, it output 1 if and only if $judge = |X \cap Y|$. For the Pricing algorithm, we simply have the Pricing output the $judge$. Note that in payable PSI, the fee is one-time. The pricing rules are determined by both parties in advance, and the server only needs to charge based on the intersection size.

We give the correctness and the security analysis of our payable PSI protocol as follows.

Correctness. We first set the output length of PRF in step 1 to $\lambda + 2\log n$, a union bound shows that the probability of collision is negligible $2^{-\lambda}$, which guarantees the intersection between X and Y is consistent with the intersection between X' and Y'. Then, for $i \in [m]$, if $X^*[i] \in Y^*[i]$, say $X^*[i] = y'_j \in Y^*[i]$, let a_q, a'_q denote the shares of $X^*[i]$, where $q = \pi^{-1}(i)$. We have $y'_j = X^*[i] = a_q \oplus a'_q$, $a_q = y'_j \oplus a'_q$. Note that $i = \pi(q)$, and the value $F_{k'}(y'_j \oplus a'_q) = F_{k'}(a_q)$ is included in set P_q. As a

[11] Note that the client in our payable PSI plays the role of the receiver in OSN.

Parameters:

- Two parties: server $\mathcal{S}$ and client $\mathcal{C}$.
- Ideal $\mathcal{F}_{\mathsf{oprf}}$ and $\mathcal{F}_{\mathsf{osn}}$ primitives specified in Figure 3, and Figure 4, respectively.

Input of $\mathcal{S}$: $X = \{x_1, \ldots, x_n\}$
Input of $\mathcal{C}$: $Y = \{y_1, \ldots, y_n\}$
Protocol:

1. The server $\mathcal{S}$ and the client $\mathcal{C}$ invoke the OPRF functionality, the client inputs Y and obtains $Y' := \{F_k(y)\}_{y \in Y}$, while the server inputs a random selected PRF key k and computes $X' := \{F_k(x)\}_{x \in X}$.
2. The server $\mathcal{S}$ inserts set X' into the Cuckoo hash table, and fills empty bins with the dummy item d. Let $m = O(n)$ denote the length of the Cuckoo hash table. $\mathcal{S}$ denotes the filled Cuckoo hash table as X^* and the item in i-th bin as $X^*[i]$ for $i \in [m]$. The client $\mathcal{C}$ inserts set Y' into the simple hash table, and deletes the duplicates in each bin, then denotes the set of items in the i-th bin as $Y^*[i]$ for $i \in [m]$.
3. $\mathcal{S}$ and $\mathcal{C}$ invokes the oblivious switching network functionality $\mathcal{F}_{\mathsf{osn}}$, the server inputs X^* and obtains $\{a_i\}_{i \in [m]}$. The client picks a random permutation π over $[m]$, inputs it to the functionality and obtains $\{a'_i\}_{i \in [m]}$. We have $a_i \oplus a'_i = X^*[\pi(i)]$ for $i \in [m]$.
4. $\mathcal{C}$ selects a random PRF key k'. Then, $\mathcal{S}$ with inputs $\{a_i\}_{i \in [m]}$ and $\mathcal{C}$ with input k' invokes another OPRF functionality. As a result, the server obtains $\{f_i\}_{i \in [m]}$, where $f_i = F_{k'}(a_i), i \in [m]$.
5. For $i \in [m]$, the client $\mathcal{C}$ computes the set $P_i := \{F_{k'}(a'_i \oplus y')\}_{y' \in Y^*[\pi(i)]}$ and pad P_i up to bin size B by different $r \leftarrow \{0,1\}^l$. Then $\mathcal{C}$ sends $\{P_i\}_{i \in [m]}$ to the server.
6. The server $\mathcal{S}$ initialize a bit string $U \in \{0,1\}^m$. Then $\mathcal{S}$ defines $U[i] = 1$ if and only if $f_i \in P_i$ and $U[i] = 0$ otherwise. The server also defines *judge* as the Hamming weight of U.
7. For $i \in [m]$, the server defines $z_i := a_i$ if $U[i] = 1$ and $z_i :=\perp$ otherwise. The server $\mathcal{S}$ sends $\{z_i\}_{i \in [m]}$ to the client.
8. The client $\mathcal{C}$ initializes $I = \emptyset$ and for $i \in [m]$:
   (a) If $z_i \neq \perp$, $\mathcal{C}$ computes $t_i := z_i \oplus a'_i$.
   (b) $\mathcal{C}$ updates the intersection $I := I \cup \{y | \exists y' \in Y^*[\pi(i)], y' = t_i, F_{k'}(y) = y'\}$.
9. The client $\mathcal{C}$ outputs the intersection I.

Fig. 10: Payable Private Set Intersection Protocol $\Pi_{\mathsf{p\text{-}psi}}$

result, the server sets $U[q] = 1$ because $F_{k'}(a_q) \in P_q$. Also, the server sets $z_q := a_q$ and sends it to the client. The client learns the PRF value of intersection element $y_j := z_q \oplus a'_q = X^*[i] \in Y^*[i]$, thus obtains the intersection element y_j corresponds to y'_j. If $X^*[i] \notin Y^*[i]$, the correctness will only be violated when $\exists F_{k'}(a_q) \in P_q$. By setting the output length of $F_{k'}(\cdot)$ to $\lambda + \log n$, a union bound shows that the probability of this event is negligible $2^{-\lambda}$. As a result, the server sets $U[i] = 0$ and sends $z_i :=\perp$ to the client. From the above analysis, the hamming weight of U is exactly the intersection cardinality, and the client will obtain all the intersection elements.

Theorem 2. *The protocol in Figure 10 along with the above* (Check, Pricing) *algorithm is a payable implement of* $\mathcal{F}_{\mathsf{psi}}$ *in the* $(\mathcal{F}_{\mathsf{oprf}}, \mathcal{F}_{\mathsf{osn}})$*-hybrid model.*

Proof. For the corrupt *semi-honest* server, we exhibit the simulator $\mathsf{Sim}_{\mathcal{S}}(X, judge)$ as follows.

1. In step 1, the simulator $\mathsf{Sim}_{\mathcal{S}}$ generates a PRF key k. Then $\mathsf{Sim}_{\mathcal{S}}$ invokes the simulator of OPRF with input k and appends the output to the view.
2. In step 2, the simulator computes X^* honestly.
3. In step 3, the simulator selects random $a_i \leftarrow \{0,1\}^l, i \in [m]$. Then $\mathsf{Sim}_{\mathcal{S}}$ invokes the simulator of OSN with input $(X^*, \{a_i\}_{i \in [m]})$ and appends the output to the view.
4. In step 4, the simulator selects random $f_i \leftarrow \{0,1\}^l, i \in [m]$. Then $\mathsf{Sim}_{\mathcal{S}}$ invokes the simulator of OPRF with input $(\{a_i\}_{i \in [m]}, \{f_i\}_{i \in [m]})$ and appends the output to the view.
5. In step 5, the simulator selects a random length m bit string U with Hamming weight *judge*. Next, if $U[i] = 0$, the simulator adds B random values to set P_i, and if $U[i] = 1$, the simulator adds $B - 1$ random values and f_i to set P_i. Finally, the simulator appends $\{P_i\}_{i \in [m]}$ to the view.

The correctness of this simulator directly follows the security of underlying protocols and the pseudorandomness of the PRF.

For the corrupt *malicious* client, we exhibit the simulator $\mathsf{Sim}_{\mathcal{R}}$ as follows:

1. In step 1, the simulator receives a set $\bar{Y}$ from the adversary. The simulator picks random values as Y' and sends it to the adversary. The simulator keeps a relationship table $R = \{(y, y')\}_{y \in \bar{Y}}$ where $y' \in Y'$ is the simulated PRF value of y.
2. In step 2, the simulator computes the cuckoo hash table Y^* corresponding to Y'.
3. In step 3, the simulator receives a permutation π from the adversary. The simulator picks random values $a'_i \leftarrow \{0,1\}^l, i \in [m]$ and sends it to the adversary.
4. In step 4, the simulator receives a PRF key k' from the adversary. The simulator also observes all the adversary's inputs to PRF and collects a list of pairs $L = \{(p_i, F_{k'}(p_i))\}$[12].
5. In step 5, the simulator receives sets $\{P_i\}_{i \in [m]}$ from the adversary and initialize a set $Y = \emptyset$. For $i \in [m], (p, F_{k'}(p)) \in L$, if $F_{k'}(p) \in P_i$, the simulator computes $y' = p \oplus a'_i$ and checks whether $y' \in Y'[\pi(i)]$. If $y' \in Y'[\pi(i)]$, the simulator updates $Y = Y \cup \{y : (y, y') \in R\}$.
6. The simulator sends Y to the ideal functionality and receives the intersection I.
7. In step 7, for $y \in Y$, if $y \in I$, the simulator finds the corresponding $y' \in Y'$. Then, the simulator finds the bin where item y' is located, w.l.o.g, $y' \in Y'[j]$ and defines $z_{\pi^{-1}(j)} = y' \oplus a'_{\pi^{-1}(j)}$. After that, the simulator sets the undefined $z_i :=\perp$ and sends $\{z_i\}_{i \in [m]}$ to the adversary.

The correctness of this simulator directly follows the security of underlying protocols and the pseudorandomness of the PRF.

6 Implementation and Performance

In this section, we experimentally evaluate our payable KPIR $\Pi_{\mathsf{p\text{-}kpir}}$ and payable PSI $\Pi_{\mathsf{p\text{-}psi}}$ protocols. The evaluation metrics are communication costs and running times.

Since there are no known protocols that achieve the payable KPIR/PSI functionality defined in Figure 7 and Figure 9, we consider using the following protocols as performance baselines and compare our protocols with them.

- Payable KPIR: Note that our payable KPIR protocol achieves both server and client privacy for a semi-honest server and a malicious client. The state-of-the-art KPIR protocols [ALP+21, MK22, AAAG22, PSY23] only achieve client privacy for a semi-honest server, and they do not protect the server privacy. To the best of our knowledge, the only similar protocol achieves malicious security is the labeled PSI protocol of Cong et al. [CMdG+21], which is equivalent to batch KPIR. Setting the size of the client's set to 1, the functionality of protocol [CMdG+21] is exactly a KPIR satisfying security against malicious client. As a result, we use the KPIR protocol [CMdG+21] as the performance baseline and compare our payable KPIR protocol with it.
- Payable PSI: Since our payable PSI protocol reveals the intersection cardinality to the server, we consider using the state-of-the-art PSI cardinality (PSI-CA) protocol as the performance benchmark. As pointed out by [GMR+21, CZZ+24], there is a huge performance gap between PSI and other private set operation protocols, even in the simplest possible cases, i.e., PSI-CA. At a high-level view, PSI-CA requires more precise control of the amount of information output in the protocol, which typically requires a more complex design. That's also why we don't use PSI protocol as a benchmark.
  We find that almost all PSI-CA protocols [HEK12, PSTY19, GMR+21, CGS22] only achieve semi-honest security. The most relevant one is the PSI-CA protocol proposed by Mingli Wu and Tsz Hon Yuen [WY23], which achieves security against malicious client. However, their protocol only considers unbalanced setting, that is, the client's set is much smaller than that of the server. The performance of their protocol will decrease significantly when both parties have the same number of items. Therefore, we consider comparing our payable PSI protocol with the state-of-the-art semi-honest PSI-CA protocol of Garimella et al. [GMR+21]. As shown in their paper, their PSI-CA protocol performs better than the circuit-based protocol [PSTY19]. As a result, we use the PSI-CA protocol [GMR+21] as the performance baseline and compare our payable PSI with it.

[12] The simulator does have this ability in the malicious secure OPRF, e.g. [JL10, RS21].

Database size n	Protocol	Value Bit Length	Setup		Online	
			Comm. (MB)	Time (s)	Comm. (MB)	Time (s)
2^{16}	KPIR [CMdG+21]	64	0.5	6.02	1.38	2.24
		128	0.5	6.33	1.62	2.44
		256	0.5	6.83	2.09	2.82
	Payable KPIR	64	0.5	8.31	1.38	2.24
		128	0.5	8.63	1.62	2.40
		256	0.5	9.11	2.09	2.83
2^{20}	KPIR [CMdG+21]	64	0.5	59.19	2.09	3.12
		128	0.5	68.51	2.68	3.76
		256	0.5	85.89	3.27	4.50
	Payable KPIR	64	0.5	99.12	2.09	3.13
		128	0.5	108.86	2.68	3.77
		256	0.5	126.65	3.86	5.05
2^{24}	KPIR [CMdG+21]	64	0.94	1198.12	3.72	14.10
		128	0.94	1577.62	5.14	20.61
		256	0.94	2295.56	7.98	33.92
	Payable KPIR	64	0.94	1890.07	3.72	14.17
		128	0.94	2256.33	5.14	20.61
		256	0.94	3014.39	8.57	34.36

Table 1: Comparisons of communication (in MB) and runtime (in seconds) between Payable KPIR and KPIR [CMdG+21] for database size ($n \in \{2^{16}, 2^{20}, 2^{24}\}$), value bit length $l \in \{64, 128, 256\}$ and 10Gbps bandwidth, 0.05ms latency.

For a fair comparison, we re-implemented their protocols on the same framework. The codes are mainly written in Java, and we use the Java Native Interface (JNI) technique to invoke SEAL library v4.1 for the homomorphic operations in the protocols. Our implementation has been open-sourced in mpc4j[13].

6.1 Experimental setup

We run our experiments on a single Intel Core i9-9900K with 3.6GHz and 128GB RAM. We simulate the network connection using Linux tc command. To better meet the potential deployment requirements, we use Netty[14] to maintain the communication channel. And we use Protocol Buffers[15] for data (de-)serialization.

We consider the following experiment setting:

- For Payable PSI protocol, we ran our experiments in both LAN and WAN settings. In the LAN setting, we set bandwidth as 10Gbps and latency as 0.05ms. For the WAN setting, we set bandwidth as 100Mbps and 10Mbps, respectively, and the latency as 80ms.
- For Payable KPIR protocol, since the communication cost is relatively small, we only executed our experiments in the LAN setting (10Gbps bandwidth and 0.05ms latency).

We divide all protocols into two phases: the one-time *Setup* phase and the *Online* phase. As the name suggests, the one-time setup phase does necessary operations before actual protocol execution, including key distribution, OPRF preprocessing, and base OT execution. The online phase does subsequent protocol executions. We emphasize that the setup phase only needs to be performed once, regardless of the number of subsequent protocol executions.

6.2 Implementation details

We set the computational security parameter $\kappa = 128$ and the statistical security parameter $\lambda = 40$. Moreover, the Stash-less Cuckoo hash is used in both our protocols and baseline protocols. All experiments use three hash functions to store n elements into $1.27n$ bins.

[13] https://https://github.com/alibaba-edu/mpc4j
[14] https://netty.io/
[15] https://developers.google.com/protocol-buffers

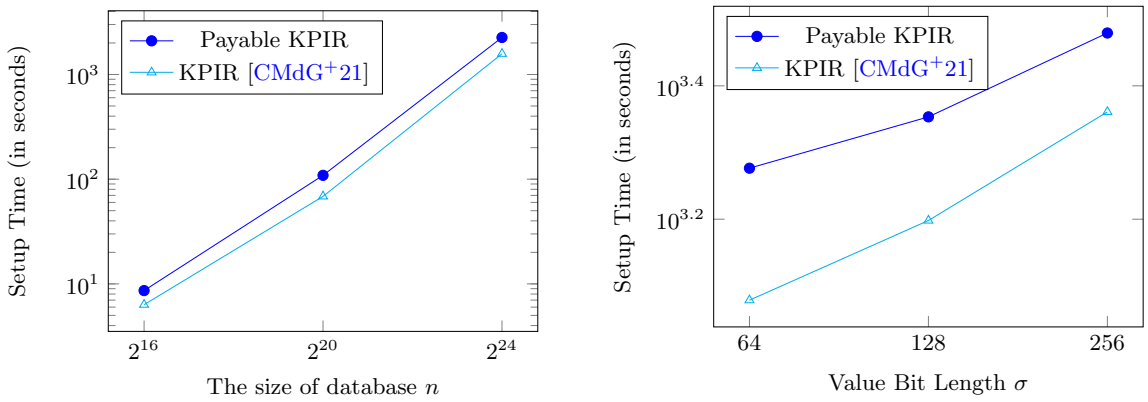

Fig. 11: Comparisons of runtime (in seconds) between Payable KPIR and KPIR [CMdG+21]. Both x and y-axis are in log scale. The first figure shows the setup runtime increases as the database size increases when the value bit length is set to 128. The second figure shows the setup runtime increases as the value bit length increases when the database size is set to 2^{24}.

Set size n	Protocol	Comm. (MB)		Time (s)											
				10Gbps				100Mbps				10Mbps			
				$T=1$		$T=8$		$T=1$		$T=8$		$T=1$		$T=8$	
		Setup	Online	Setup	Online	Setup	Online	Setup	Online	Setup	Online	Setup	Online	Setup	Online
2^8	PSI-CA [GMR+21]	0.03	0.15	0.06	0.03	0.03	0.06	0.95	0.69	0.72	0.82	1.01	0.68	0.75	0.86
	Payable PSI	0	0.28	0.02	0.16	0.01	0.09	0.19	3.70	0.18	3.48	0.19	3.73	0.18	3.68
2^{12}	PSI-CA [GMR+21]	0.03	2.96	0.05	0.26	0.02	0.19	0.81	1.06	0.70	1.74	0.81	3.42	0.71	3.95
	Payable PSI	0	5.19	0.03	1.61	0.01	0.49	0.19	6.79	0.18	6.19	0.21	10.66	0.18	9.79
2^{16}	PSI-CA [GMR+21]	0.03	56.61	0.05	4.36	0.02	2.23	0.80	11.33	0.70	9.10	0.83	55.55	0.72	53.73
	Payable PSI	0	103.32	0.02	25.10	0.01	6.07	0.22	39.93	0.17	22.78	0.22	113.77	0.18	101.55
2^{20}	PSI-CA [GMR+21]	0.03	1056.30	0.05	94.67	0.02	45.77	0.80	195.25	0.70	152.55	0.82	1019.21	0.72	979.17
	Payable PSI	0	1988.92	0.02	424.36	0.01	143.55	0.19	578.65	0.17	320.73	0.22	2021.27	0.18	1840.90

Table 2: Comparisons of communication (in MB) and runtime (in seconds) between Payable PSI and PSI-CA [GMR+21] for set size ($n \in \{2^8, 2^{12}, 2^{16}, 2^{20}\}$), thread ($T \in \{1, 8\}$) and bandwidth ($\{10\text{Gbps}, 100\text{Mbps}, 10\text{Mbps}\}$)

- Payable KPIR. In our payable KPIR protocol and the baseline protocol [CMdG+21], we mainly need to implement the KPIR protocol. To fully re-implement the protocol in our framework, we study the implementation details and follow the parameter selections of the open-source implementations of KPIR protocol.
- Payable PSI. To enable our payable PSI protocol and the baseline protocols, we implement the following components.
  - OPRF. We implement the OPRF protocol introduced in [RA18] as our sub-protocol. For this OPRF protocol, the sender can randomly generate an OPRF key in advance, which is independent of the receiver's input. For the sender's input, it can hash each item to a uniformly random elliptic curve group element (we treat the hash as a random oracle) and multiply this group element with the sampled key. Furthermore, we use the FourQ elliptic curve [CL15] to speed up the scalar multiplication operations.
  - OSN. We choose the state-of-the-art OSN protocol introduced in [MS13] as our sub-protocol. Moreover, to satisfy the security proof of our Payable PSI protocol, the underlying COT protocol should be actively secure, and we implement the COT protocol based on [KOS15]. However, the OSN protocol used in the baseline protocol [GMR+21] does not need to be actively secure, a passive secure COT protocol is enough, and we implement the COT protocol based on [ALSZ13].

6.3 Payable KPIR

As we mentioned before, we compare our payable KPIR protocol with the KPIR protocol in [CMdG+21] as a performance baseline. We report detailed comparisons in Table 1 and Figure 11 for database sizes $n = \{2^{16}, 2^{20}, 2^{24}\}$ and bit-length of the data value $l = \{64, 128, 256\}$.

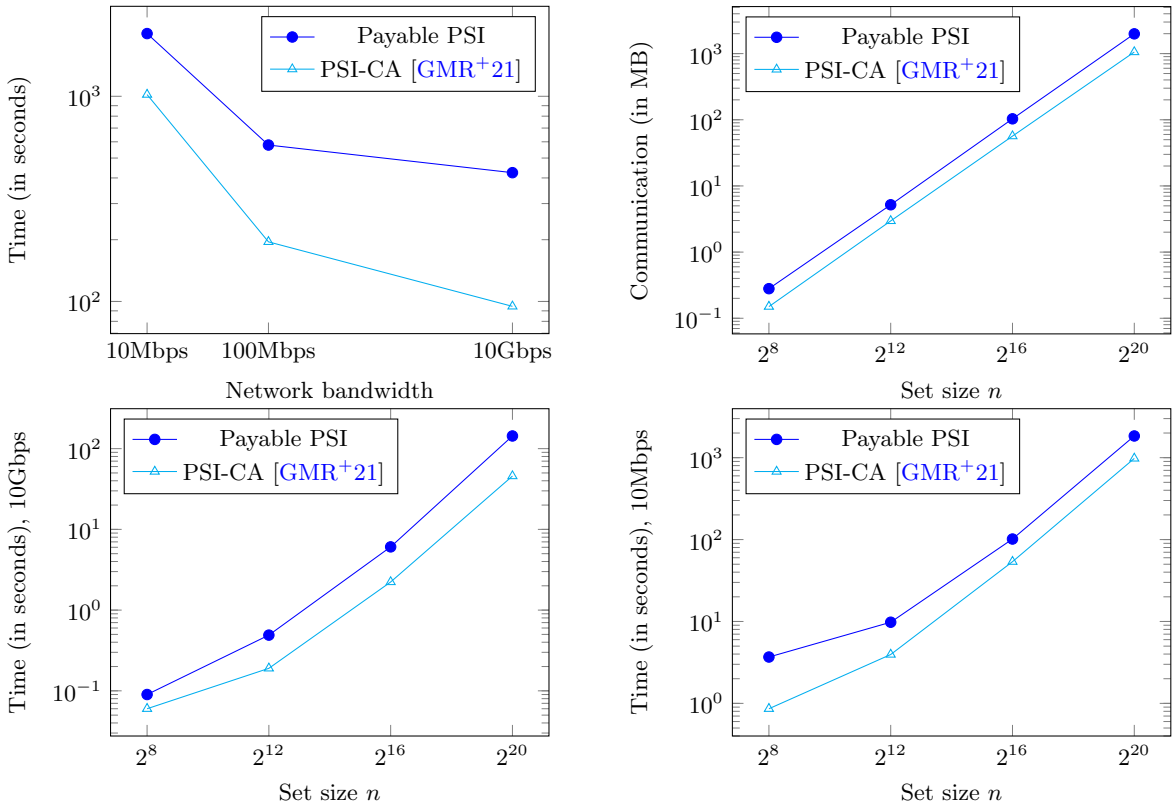

Fig. 12: Comparisons of communication (in MB) and runtime (in seconds) between Payable PSI and PSI-CA [GMR+21]. Both x and y-axis are in log scale. The first figure shows the runtime decreases as the bandwidth increases. The second figure shows the communication cost increases as the set size increases. The last two figures show the runtime increases as the set size increases in the 10Gbps and 10Mbps settings, respectively.

As shown in Table 1 and Figure 11, our payable KPIR protocols do not introduce too much overhead over KPIR [CMdG+21]. Note that our payable KPIR needs to execute an OPRF to preprocess the database before invoking the underlining KPIR [CMdG+21]. This is reflected in the table that the setup runtime of our payable KPIR is about $1.3 - 1.6\times$ slower than that of the KPIR protocol [CMdG+21]. In the online stage, the second OPRF in our payable KPIR is extremely fast, resulting in the online communication and runtime between payable KPIR and KPIR [CMdG+21] are almost the same.

We also test the variation of runtime and communication with the bit length of the database value. Note that the underlying KPIR [CMdG+21] uses fully homomorphic encryption (FHE) scheme [BGV12, FV12] as the building block, and the plaintext space of FHE is relatively small. For long database values, we need to divide the values into multiple small blocks. The longer the data value, the more blocks it has, leading to decreased protocol performance.

6.4 Payable PSI

As we mentioned before, we compare our payable PSI protocol with the PSI-CA protocol in [GMR+21] as a performance baseline. We report detailed comparisons in Table 2 and Figure 12 for set sizes $n = \{2^8, 2^{12}, 2^{16}, 2^{20}\}$, the number of threads $T = \{1, 8\}$ and bandwidth in $\{10\text{Gbps}, 100\text{Mbps}, 10\text{Mbps}\}$.

The communication of our payable PSI protocol is about twice larger than the PSI-CA protocol [GMR+21], mainly due to the need for OPRF preprocessing in our protocol. In terms of runtime, in LAN and single thread setting, our protocol is $4.5 - 6.2\times$ slower than PSI-CA. Since our protocol is very amenable to parallelization, when increasing T from 1 to 8, the performance gap between our protocol and PSI-CA has narrowed to $1.5 - 3.2\times$. As the bandwidth decreases, the gap between our protocol and PSI-CA protocol also narrows gradually. Specifically, when the bandwidth is 10Mbps, the communication cost becomes a bottleneck. The performance ratio between our protocol and PSI-CA is about $2\times$, which is consistent with the communication cost ratio.

Note that our payable PSI protocol achieves security against malicious client, and it is more powerful in the data pricing applications. While the PSI-CA protocol [GMR+21] only achieves semi-honest security, and it does not support the client to obtain the intersection. The performance gap between our payable PSI and PSI-CA [GMR+21] is quite mild.

7 Conclusions

In this work, we proposed the concept of Payable Secure Computation (PSC) and designed payable protocols for two specific scenarios: Keyword Private Information Retrieval (KPIR) and Private Set Intersection (PSI). The experiments show that our protocols do not increase the cost much compared to protocols that do not support pricing.

Due to the rapid development of the data market, our current construction is still relatively simple. Considering more complex scenarios, such as the support for batch queries in KPIR, the server may want to charge based on the range of query data, or in PSI, the server may want to attach a weight to each item, and the pricing is the sum of the weights in the intersection. The PSC framework could be adapted to support richer query types (e.g., range queries, aggregations) by combining the existing OPRF-based checks with verifiable computation techniques (e.g., zk-SNARKs or authenticated data structures). Payment logic might be tied to incremental results (e.g., per-row decryption in private database queries).

Extending the PSC to multi-party setting is also a valuable future work, corresponding to the scenario where multiple servers jointly provide data in the data market, and multiple clients purchase different data. In the multiparty setting, two properties must be considered. The first is fair payment distribution: Splitting rewards among servers based on contribution (e.g., using threshold OPRFs). The second is joint accountability: Clients could pay only if a threshold of servers provide valid outputs, deterring collusion.

References

[AAAG22] Ishtiyaque Ahmad, Divyakant Agrawal, Amr El Abbadi, and Trinabh Gupta. Pantheon: Private retrieval from public key-value store. *Proc. VLDB Endow.*, 16(4):643–656, 2022.

[ACLS18] Sebastian Angel, Hao Chen, Kim Laine, and Srinath T. V. Setty. PIR with compressed queries and amortized query processing. In *2018 IEEE Symposium on Security and Privacy, SP 2018, Proceedings, 21-23 May 2018, San Francisco, California, USA*, pages 962–979. IEEE Computer Society, 2018.

[ALP+21] Asra Ali, Tancrède Lepoint, Sarvar Patel, Mariana Raykova, Phillipp Schoppmann, Karn Seth, and Kevin Yeo. Communication-computation trade-offs in PIR. In Michael Bailey and Rachel Greenstadt, editors, *30th USENIX Security Symposium, USENIX Security 2021, August 11-13, 2021*, pages 1811–1828. USENIX Association, 2021.

[ALSZ13] Gilad Asharov, Yehuda Lindell, Thomas Schneider, and Michael Zohner. More efficient oblivious transfer and extensions for faster secure computation. In *CCS 2013*, 2013.

[aws] awsmarket[n.d.]. Aws data exchange. https://aws.amazon.com/data-exchange/.

[BG12] Stephanie Bayer and Jens Groth. Efficient zero-knowledge argument for correctness of a shuffle. In David Pointcheval and Thomas Johansson, editors, *Advances in Cryptology - EUROCRYPT 2012 - 31st Annual International Conference on the Theory and Applications of Cryptographic Techniques, Cambridge, UK, April 15-19, 2012. Proceedings*, volume 7237 of *Lecture Notes in Computer Science*, pages 263–280. Springer, 2012.

[BGV12] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (leveled) fully homomorphic encryption without bootstrapping. In Shafi Goldwasser, editor, *Innovations in Theoretical Computer Science 2012, Cambridge, MA, USA, January 8-10, 2012*, pages 309–325. ACM, 2012.

[BR93] Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In Dorothy E. Denning, Raymond Pyle, Ravi Ganesan, Ravi S. Sandhu, and Victoria Ashby, editors, *CCS '93, Proceedings of the 1st ACM Conference on Computer and Communications Security, Fairfax, Virginia, USA, November 3-5, 1993*, pages 62–73. ACM, 1993.

[CGN98] Benny Chor, Niv Gilboa, and Moni Naor. Private information retrieval by keywords. *IACR Cryptol. ePrint Arch.*, page 3, 1998.

[CGS22] Nishanth Chandran, Divya Gupta, and Akash Shah. Circuit-psi with linear complexity via relaxed batch OPPRF. *Proc. Priv. Enhancing Technol.*, 2022(1):353–372, 2022.

[CGT12] Emiliano De Cristofaro, Paolo Gasti, and Gene Tsudik. Fast and private computation of cardinality of set intersection and union. In Josef Pieprzyk, Ahmad-Reza Sadeghi, and Mark Manulis, editors, *Cryptology and Network Security, 11th International Conference, CANS 2012, Darmstadt, Germany, December 12-14, 2012. Proceedings*, 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 David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang, editors, *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS 2018, Toronto, ON, Canada, October 15-19, 2018*, pages 1223–1237. ACM, 2018.

[CILO22] Wutichai Chongchitmate, Yuval Ishai, Steve Lu, and Rafail Ostrovsky. PSI from ring-ole. In Heng Yin, Angelos Stavrou, Cas Cremers, and Elaine Shi, editors, *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security, CCS 2022, Los Angeles, CA, USA, November 7-11, 2022*, pages 531–545. ACM, 2022.

[CKGS98] Benny Chor, Eyal Kushilevitz, Oded Goldreich, and Madhu Sudan. Private information retrieval. *J. ACM*, 45(6):965–981, 1998.

[CKK17] Lingjiao Chen, Paraschos Koutris, and Arun Kumar. Model-based pricing: Do not pay for more than what you learn! In Sebastian Schelter and Reza Zadeh, editors, *Proceedings of the 1st Workshop on Data Management for End-to-End Machine Learning, DEEM@SIGMOD 2017, Chicago, IL, USA, May 14, 2017*, pages 1:1–1:4. ACM, 2017.

[CKK19] Lingjiao Chen, Paraschos Koutris, and Arun Kumar. Towards model-based pricing for machine learning in a data marketplace. In Peter A. Boncz, Stefan Manegold, Anastasia Ailamaki, Amol Deshpande, and Tim Kraska, editors, *Proceedings of the 2019 International Conference on Management of Data, SIGMOD Conference 2019, Amsterdam, The Netherlands, June 30 - July 5, 2019*, pages 1535–1552. ACM, 2019.

[CKT10] Emiliano De Cristofaro, Jihye Kim, and Gene Tsudik. Linear-complexity private set intersection protocols secure in malicious model. In Masayuki Abe, editor, *Advances in Cryptology - ASIACRYPT 2010*, volume 6477 of *Lecture Notes in Computer Science*, pages 213–231. Springer, 2010.

[CL15] Craig Costello and Patrick Longa. Four$\mathbb{Q}$: Four-dimensional decompositions on a $\mathbb{Q}$-curve over the mersenne prime. In Tetsu Iwata and Jung Hee Cheon, editors, *Advances in Cryptology - ASIACRYPT 2015 - 21st International Conference on the Theory and Application of Cryptology and Information Security, Auckland, New Zealand, November 29 - December 3, 2015, Proceedings, Part I*, volume 9452 of *Lecture Notes in Computer Science*, pages 214–235. Springer, 2015.

[CM20] Melissa Chase and Peihan Miao. Private set intersection in the internet setting from lightweight oblivious PRF. In *CRYPTO 2020*, 2020.

[CMdG+21] Kelong Cong, Radames Cruz Moreno, Mariana Botelho da Gama, Wei Dai, Ilia Iliashenko, Kim Laine, and Michael Rosenberg. Labeled PSI from homomorphic encryption with reduced computation and communication. In Yongdae Kim, Jong Kim, Giovanni Vigna, and Elaine Shi, editors, *CCS '21: 2021 ACM SIGSAC Conference on Computer and Communications Security, Virtual Event, Republic of Korea, November 15 - 19, 2021*, pages 1135–1150. ACM, 2021.

[CS03] Jan Camenisch and Victor Shoup. Practical verifiable encryption and decryption of discrete logarithms. In Dan Boneh, editor, *Advances in Cryptology - CRYPTO 2003, 23rd Annual International Cryptology Conference, Santa Barbara, California, USA, August 17-21, 2003, Proceedings*, volume 2729 of *Lecture Notes in Computer Science*, pages 126–144. Springer, 2003.

[CYW+22] Chen Chen, Ye Yuan, Zhenyu Wen, Guoren Wang, and Anteng Li. GQP: A framework for scalable and effective graph query-based pricing. In *38th IEEE International Conference on Data Engineering, ICDE 2022, Kuala Lumpur, Malaysia, May 9-12, 2022*, pages 1573–1585. IEEE, 2022.

[CZZ+24] Yu Chen, Min Zhang, Cong Zhang, Minglang Dong, and Weiran Liu. Private set operations from multi-query reverse private membership test. In Qiang Tang and Vanessa Teague, editors, *Public-Key Cryptography - PKC 2024 - 27th IACR International Conference on Practice and Theory of Public-Key Cryptography, Sydney, NSW, Australia, April 15-17, 2024, Proceedings, Part III*, volume 14603 of *Lecture Notes in Computer Science*, pages 387–416. Springer, 2024.

[DK17] Shaleen Deep and Paraschos Koutris. QIRANA: A framework for scalable query pricing. In Semih Salihoglu, Wenchao Zhou, Rada Chirkova, Jun Yang, and Dan Suciu, editors, *Proceedings of the 2017 ACM International Conference on Management of Data, SIGMOD Conference 2017,*

Chicago, IL, USA, May 14-19, 2017, pages 699–713. ACM, 2017.

[DMRY09] Dana Dachman-Soled, Tal Malkin, Mariana Raykova, and Moti Yung. Efficient robust private set intersection. In Michel Abdalla, David Pointcheval, Pierre-Alain Fouque, and Damien Vergnaud, editors, *Applied Cryptography and Network Security, 7th International Conference, ACNS 2009, Paris-Rocquencourt, France, June 2-5, 2009. Proceedings*, volume 5536 of *Lecture Notes in Computer Science*, pages 125–142, 2009.

[DPC23] Alex Davidson, Gonçalo Pestana, and Sofía Celi. Frodopir: Simple, scalable, single-server private information retrieval. *Proc. Priv. Enhancing Technol.*, 2023(1):365–383, 2023.

[DY05] Yevgeniy Dodis and Aleksandr Yampolskiy. A verifiable random function with short proofs and keys. In Serge Vaudenay, editor, *Public Key Cryptography - PKC 2005, 8th International Workshop on Theory and Practice in Public Key Cryptography, Les Diablerets, Switzerland, January 23-26, 2005, Proceedings*, volume 3386 of *Lecture Notes in Computer Science*, pages 416–431. Springer, 2005.

[Fer22] Raul Castro Fernandez. Protecting data markets from strategic buyers. In Zachary G. Ives, Angela Bonifati, and Amr El Abbadi, editors, *SIGMOD '22: International Conference on Management of Data, Philadelphia, PA, USA, June 12 - 17, 2022*, pages 1755–1769. ACM, 2022.

[FHNP16] Michael J. Freedman, Carmit Hazay, Kobbi Nissim, and Benny Pinkas. Efficient set intersection with simulation-based security. *J. Cryptology*, 29(1):115–155, 2016.

[FIPR05] Michael J. Freedman, Yuval Ishai, Benny Pinkas, and Omer Reingold. Keyword search and oblivious pseudorandom functions. In Joe Kilian, editor, *Theory of Cryptography, Second Theory of Cryptography Conference, TCC 2005, Cambridge, MA, USA, February 10-12, 2005, Proceedings*, volume 3378 of *Lecture Notes in Computer Science*, pages 303–324. Springer, 2005.

[FNP04] Michael J. Freedman, Kobbi Nissim, and Benny Pinkas. Efficient private matching and set intersection. In *EUROCRYPT 2004*, 2004.

[FSF20] Raul Castro Fernandez, Pranav Subramaniam, and Michael J. Franklin. Data market platforms: Trading data assets to solve data problems. *Proc. VLDB Endow.*, 13(11):1933–1947, 2020.

[FV12] Junfeng Fan and Frederik Vercauteren. Somewhat practical fully homomorphic encryption. *IACR Cryptol. ePrint Arch.*, page 144, 2012.

[Gam84] Taher El Gamal. A public key cryptosystem and a signature scheme based on discrete logarithms. In G. R. Blakley and David Chaum, editors, *Advances in Cryptology, Proceedings of CRYPTO '84, Santa Barbara, California, USA, August 19-22, 1984, Proceedings*, volume 196 of *Lecture Notes in Computer Science*, pages 10–18. Springer, 1984.

[GIKM00] Yael Gertner, Yuval Ishai, Eyal Kushilevitz, and Tal Malkin. Protecting data privacy in private information retrieval schemes. *J. Comput. Syst. Sci.*, 60(3):592–629, 2000.

[GMR+21] Gayathri Garimella, Payman Mohassel, Mike Rosulek, Saeed Sadeghian, and Jaspal Singh. Private set operations from oblivious switching. In *PKC 2021*, 2021.

[GMW87] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game or A completeness theorem for protocols with honest majority. In *STOC 1987*, 1987.

[Gol04] Oded Goldreich. *The Foundations of Cryptography - Volume 2: Basic Applications*. Cambridge University Press, 2004.

[GPR+21] Gayathri Garimella, Benny Pinkas, Mike Rosulek, Ni Trieu, and Avishay Yanai. Oblivious key-value stores and amplification for private set intersection. In *CRYPTO 2021*, 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, San Diego, California, USA, February 5-8, 2012*. The Internet Society, 2012.

[HFH99] Bernardo A. Huberman, Matthew K. Franklin, and Tad Hogg. Enhancing privacy and trust in electronic communities. In Stuart I. Feldman and Michael P. Wellman, editors, *Proceedings of the First ACM Conference on Electronic Commerce (EC-99), Denver, CO, USA, November 3-5, 1999*, pages 78–86. ACM, 1999.

[HHC+23] Alexandra Henzinger, Matthew M. Hong, Henry Corrigan-Gibbs, Sarah Meiklejohn, and Vinod Vaikuntanathan. One server for the price of two: Simple and fast single-server private information retrieval. In Joseph A. Calandrino and Carmela Troncoso, editors, *32nd USENIX Security Symposium, USENIX Security 2023, Anaheim, CA, USA, August 9-11, 2023*. USENIX Association, 2023.

[IKNP03] Yuval Ishai, Joe Kilian, Kobbi Nissim, and Erez Petrank. Extending oblivious transfers efficiently. In *CRYPTO 2003*, 2003.

[JL10] Stanislaw Jarecki and Xiaomin Liu. Fast secure computation of set intersection. In Juan A. Garay and Roberto De Prisco, editors, *Security and Cryptography for Networks, 7th International Conference, SCN 2010, Amalfi, Italy, September 13-15, 2010. Proceedings*, volume 6280 of *Lecture Notes in Computer Science*, pages 418–435. Springer, 2010.

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

[KKRT16] Vladimir Kolesnikov, Ranjit Kumaresan, Mike Rosulek, and Ni Trieu. Efficient batched oblivious PRF with applications to private set intersection. In *CCS 2016*, 2016.

[KOS15] Marcel Keller, Emmanuela Orsini, and Peter Scholl. Actively secure OT extension with optimal overhead. In Rosario Gennaro and Matthew Robshaw, editors, *Advances in Cryptology - CRYPTO 2015 - 35th Annual Cryptology Conference, Santa Barbara, CA, USA, August 16-20, 2015, Proceedings, Part I*, volume 9215 of *Lecture Notes in Computer Science*, pages 724–741. Springer, 2015.

[KUB+12a] Paraschos Koutris, Prasang Upadhyaya, Magdalena Balazinska, Bill Howe, and Dan Suciu. Query-based data pricing. In Michael Benedikt, Markus Krötzsch, and Maurizio Lenzerini, editors, *Proceedings of the 31st ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS 2012, Scottsdale, AZ, USA, May 20-24, 2012*, pages 167–178. ACM, 2012.

[KUB+12b] Paraschos Koutris, Prasang Upadhyaya, Magdalena Balazinska, Bill Howe, and Dan Suciu. Querymarket demonstration: Pricing for online data markets. *Proc. VLDB Endow.*, 5(12):1962–1965, 2012.

[KUB+13] Paraschos Koutris, Prasang Upadhyaya, Magdalena Balazinska, Bill Howe, and Dan Suciu. Toward practical query pricing with querymarket. In Kenneth A. Ross, Divesh Srivastava, and Dimitris Papadias, editors, *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2013, New York, NY, USA, June 22-27, 2013*, pages 613–624. ACM, 2013.

[KUB+15] Paraschos Koutris, Prasang Upadhyaya, Magdalena Balazinska, Bill Howe, and Dan Suciu. Query-based data pricing. *J. ACM*, 62(5):43:1–43:44, 2015.

[Lin17] Yehuda Lindell. How to simulate it - A tutorial on the simulation proof technique. In Yehuda Lindell, editor, *Tutorials on the Foundations of Cryptography*, pages 277–346. Springer International Publishing, 2017.

[LK14] Bing-Rong Lin and Daniel Kifer. On arbitrage-free pricing for general data queries. *Proc. VLDB Endow.*, 7(9):757–768, 2014.

[MAP12] M. K. Mohan Murthy, Sanjay Harogolige Adimurthy, and Ashwini Janagal Padmanabha. Pricing models and pricing schemes of iaas providers: a comparison study. In Kaliappan Gopalan and Sabu M. Thampi, editors, *2012 International Conference on Advances in Computing, Communications and Informatics, ICACCI '12, Chennai, India, August 3-5, 2012*, pages 143–147. ACM, 2012.

[MBFK16] Carlos Aguilar Melchor, Joris Barrier, Laurent Fousse, and Marc-Olivier Killijian. XPIR : Private information retrieval for everyone. *Proc. Priv. Enhancing Technol.*, 2016(2):155–174, 2016.

[MCR21] Muhammad Haris Mughees, Hao Chen, and Ling Ren. Onionpir: Response efficient single-server PIR. In Yongdae Kim, Jong Kim, Giovanni Vigna, and Elaine Shi, editors, *CCS '21: 2021 ACM SIGSAC Conference on Computer and Communications Security, Virtual Event, Republic of Korea, November 15 - 19, 2021*, pages 2292–2306. ACM, 2021.

[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, Oakland, California, USA, April 7-9, 1986*, pages 134–137. IEEE Computer Society, 1986.

[MK22] Rasoul Akhavan Mahdavi and Florian Kerschbaum. Constant-weight PIR: single-round keyword PIR via constant-weight equality operators. In Kevin R. B. Butler and Kurt Thomas, editors, *31st USENIX Security Symposium, USENIX Security 2022, Boston, MA, USA, August 10-12, 2022*, pages 1723–1740. USENIX Association, 2022.

[MPR+20] Peihan Miao, Sarvar Patel, Mariana Raykova, Karn Seth, and Moti Yung. Two-sided malicious security for private intersection-sum with cardinality. In Daniele Micciancio and Thomas Ristenpart, editors, *Advances in Cryptology - CRYPTO 2020 - 40th Annual International Cryptology Conference, CRYPTO 2020, Santa Barbara, CA, USA, August 17-21, 2020, Proceedings, Part III*, volume 12172 of *Lecture Notes in Computer Science*, pages 3–33. Springer, 2020.

[MS13] Payman Mohassel and Seyed Saeed Sadeghian. How to hide circuits in MPC an efficient framework for private function evaluation. In *EUROCRYPT 2013*, 2013.

[MW22] Samir Jordan Menon and David J. Wu. Spiral: Fast, high-rate single-server pir via fhe composition. In *2022 IEEE Symposium on Security and Privacy (SP)*, pages 930–947, 2022.

[Ped91] Torben P. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In Joan Feigenbaum, editor, *Advances in Cryptology - CRYPTO '91, 11th Annual International Cryptology Conference, Santa Barbara, California, USA, August 11-15, 1991, Proceedings*, volume 576 of *Lecture Notes in Computer Science*, pages 129–140. Springer, 1991.

[PR04] Rasmus Pagh and Flemming Friche Rodler. Cuckoo hashing. *J. Algorithms*, 51(2):122–144, 2004.

[PRTY19] Benny Pinkas, Mike Rosulek, Ni Trieu, and Avishay Yanai. Spot-light: Lightweight private set intersection from sparse OT extension. In *CRYPTO 2019*, 2019.

[PSTY19] Benny Pinkas, Thomas Schneider, Oleksandr Tkachenko, and Avishay Yanai. Efficient circuit-based PSI with linear communication. In Yuval Ishai and Vincent Rijmen, editors, *Advances in Cryptology - EUROCRYPT 2019, Part III*, volume 11478 of *LNCS*, pages 122–153. Springer, 2019.

[PSY23] Sarvar Patel, Joon Young Seo, and Kevin Yeo. Don't be dense: Efficient keyword PIR for sparse databases. In Joseph A. Calandrino and Carmela Troncoso, editors, *32nd USENIX Security Symposium, USENIX Security 2023, Anaheim, CA, USA, August 9-11, 2023*. USENIX Association, 2023.

[RA18] Amanda Cristina Davi Resende and Diego F. Aranha. Faster unbalanced private set intersection. In Sarah Meiklejohn and Kazue Sako, editors, *Financial Cryptography and Data Security - 22nd International Conference, FC 2018, Nieuwpoort, Curaçao, February 26 - March 2, 2018, Revised Selected Papers*, volume 10957 of *Lecture Notes in Computer Science*, pages 203–221. Springer, 2018.

[RR22] Srinivasan Raghuraman and Peter Rindal. Blazing fast PSI from improved OKVS and subfield VOLE. In Heng Yin, Angelos Stavrou, Cas Cremers, and Elaine Shi, editors, *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security, CCS 2022, Los Angeles, CA, USA, November 7-11, 2022*, pages 2505–2517. ACM, 2022.

[RS21] Peter Rindal and Phillipp Schoppmann. VOLE-PSI: fast OPRF and circuit-psi from vector-ole. In *EUROCRYPT 2021*, 2021.

[RT21] Mike Rosulek and Ni Trieu. Compact and malicious private set intersection for small sets. In Yongdae Kim, Jong Kim, Giovanni Vigna, and Elaine Shi, editors, *CCS '21: 2021 ACM SIGSAC Conference on Computer and Communications Security, Virtual Event, Republic of Korea, November 15 - 19, 2021*, pages 1166–1181. ACM, 2021.

[TCLZ23] Binbin Tu, Yu Chen, Qi Liu, and Cong Zhang. Fast unbalanced private set union from fully homomorphic encryption. In *CCS 2023 (To appear)*, 2023.

[vdSDLP19] Stephanie van de Sandt, Sünje Dallmeier-Tiessen, Artemis Lavasa, and Vivien Petras. The definition of reuse. *Data Sci. J.*, 18:22, 2019.

[WY23] Mingli Wu and Tsz Hon Yuen. Efficient unbalanced private set intersection cardinality and user-friendly privacy-preserving contact tracing. In *32nd USENIX Security Symposium (USENIX Security 23)*, pages 283–300, Anaheim, CA, August 2023. USENIX Association.

[Yao86] Andrew Chi-Chih Yao. How to generate and exchange secrets (extended abstract). In *FOCS*, 1986.

[ZCL+23] Cong Zhang, Yu Chen, Weiran Liu, Min Zhang, and Dongdai Lin. Linear private set union from Multi-Query reverse private membership test. In *32nd USENIX Security Symposium (USENIX Security 23)*, pages 337–354, Anaheim, CA, August 2023. USENIX Association.

A Existing Multi-Query Reverse Private Membership Test Protocols

We investigated the existing mq-RPMT constructions. To the best of our knowledge, all known mq-RPMT constructions only satisfy semi-honest security. Here we list these constructions and briefly discuss why they are not secure against malicious clients.

A.1 Mq-RPMT of Jia et al. [JSZ+22]

Jia et al. [JSZ+22] proposed two PSU protocols. One is based on permuted mq-PMT, and the other is based on mq-RPMT. The main difference is the definition of indication set P_i. In permuted mq-PMT, P_i is defined as the PRF values of a'_i XOR some y's, that is, a'_i is fixed. Since a'_i is the share of $x_{\pi(i)}$, the server obtains the result of whether $x_{\pi(i)} \in Y$. In mq-RPMT, the role of y and a' is reversed, P_i is defined as the PRF values of y_i XOR some a's, that is, y_i is fixed. As a result, the server will learn whether $y_i \in X$ from P_i. We give the mq-RPMT protocol of [JSZ+22] in Figure 13.

Parameters:

- Two parties: server $\mathcal{S}$ and client $\mathcal{C}$.
- Ideal $\mathcal{F}_{\mathsf{oprf}}$ and $\mathcal{F}_{\mathsf{osn}}$ primitives specified in Figure 3, and Figure 4, respectively.

Input of $\mathcal{S}$: $X = \{x_1, \ldots, x_n\}$
Input of $\mathcal{C}$: $Y = \{y_1, \ldots, y_n\}$
Protocol:

1. The server $\mathcal{S}$ inserts set X into the Cuckoo hash table with hash functions $H = \{h_1, \ldots, h_\gamma\}$, and fills empty bins with the dummy item d, then denotes the filled Cuckoo hash table as X^* and the item in i-th bin as $X^*[i]$. The client $\mathcal{C}$ inserts set Y into the simple hash table with the same hash functions H, and deletes the duplicates in each bin, then denotes the set of items in the i-th bin as $Y^*[i]$.
2. $\mathcal{S}$ and $\mathcal{C}$ invokes the oblivious switching network functionality, the server inputs X^* and obtains $\{a_i\}_{i\in[m]}$. The client picks a random permutation over $[m]$, inputs it to the functionality and obtains $\{a'_i\}_{i\in[m]}$. We have $a_i \oplus a'_i = x_{\pi(i)}$ for $i \in [m]$.
3. $\mathcal{S}$ with inputs $\{a_i\}_{i\in[m]}$ and $\mathcal{C}$ invokes another OPRF functionality. As a result, the server obtains $\{f_i\}_{i\in[m]}$, where $f_i = F_k(a_i)$, and the client obtains a PRF key k.
4. For $i \in [m]$, the client $\mathcal{C}$ defines $Q_i := \{\pi^{-1}(h_1(y_i)), \ldots, \pi^{-1}(h_\gamma(y_i))\}$, computes the set $P_i := \{F_k(y_i \oplus a'_j)\}_{j\in Q_i}$ and pad P_i up to bin size B by different $r \leftarrow \{0,1\}^l$. Then $\mathcal{C}$ sends $\{P_i\}_{i\in[m]}$ to the server.
5. The server $\mathcal{S}$ initialize a bit string $U \in \{0,1\}^m$. Then $\mathcal{S}$ defines $U[i] = 1$ if and only if $F_k(a_i) \in P_i$ and $U[i] = 0$ otherwise.

Fig. 13: mq-RPMT protocol of [JSZ+22]

Security against Malicious Client. As we discussed above, the difference between permuted mq-PMT and mq-RPMT is the definition of P_i. This difference may cause a malicious client to perform a *selective failure attack* in mq-RPMT. Since a'_j travels over all positions corresponding to the cuckoo hashing table of X, a malicious client can remove some $F_k(y_i \oplus a'_j)$ in P_i. Then, if the client learns an intersection element x in i-th bin, it knows Cuckoo hashing positions of x, which may reveal information about the server's entire input set. There is no such issue in permuted mq-PMT because in permuted mq-PMT, a_i is fixed, and y is traversal. Removing some y will only result in the simulator not being able to extract the input y, which means that the adversary has removed the element y from its own set. This attack is trivial since the adversary could change its input arbitrarily.

A.2 Mq-RPMT of Zhang et al. [ZCL+23]

Zhang et al. [ZCL+23] proposed two mq-RPMT constructions, one is based on Symmetric Key Encryption (SKE) and general two Party Computation (2PC), and the other is based on Rerandomizable

Public Key Encryption (Rerand-PKE). We note that their Rerand-PKE based construction may leak some information about the elements that are not in the intersection[16]. Therefore, we focus on their SKE-based construction.

The construction of [ZCL+23] uses a functionality called Vector Oblivious Decryption-then-Matching (VODM). This functionality receives a vector of ciphertexts $\{c_i\}_{i\in[n]}$ from the sender, a key k, and a plaintext m from the receiver. The fuctionality sends a bit string $b \in \{0,1\}^n$ to the receiver, the i-th bit $b_i = 1$ if and only if $\mathsf{Dec}_k(c_i) = m$. We give the formal definition of VODM in Figure 14. They also use an Oblivious Key-Value Store (OKVS) scheme [GPR+21]. Simply speaking, OKVS is a data structure that maps a set of keys to corresponding values. It consists of two algorithm $(\mathsf{Encode}, \mathsf{Decode})$. The Encode algorithm takes a set of key-value pairs $\{(x_i, y_i)\}_{i\in[n]}$ as input, and outputs an object D. The Decode algorithm takes D and a key x as input and outputs a value y. The correctness requires that if (x, y) is an input to generate D, then, $\mathsf{Decode}(D, x) = y$. We give the mq-RPMT protocol of [ZCL+23] in Figure 15.

Parameters: Sender $\mathcal{S}$, Receiver $\mathcal{R}$, set sizes n, an encryption scheme $\mathcal{E} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$.
Functionality:

- Wait for input k and s from the receiver $\mathcal{R}$.
- Wait for input $\{s_1^*, \ldots, s_n^*\} \subset \{0,1\}^*$ from the sender $\mathcal{S}$.
- For $i \in [n]$:
  Compute $s_i' = \mathsf{Dec}(k, s_i^*)$. If $s_i' = s$, let $b_i = 1$, otherwise $b_i = 0$.
- Give output $b \in \{0,1\}^n$ to the receiver $\mathcal{R}$.

Fig. 14: Vector Oblivious Decryption-then-Matching Functionality $\mathcal{F}_{\mathsf{vodm}}$

Parameters:

- Two parties: server $\mathcal{S}$ and client $\mathcal{C}$.
- An SKE scheme $\mathcal{E} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$.
- Ideal $\mathcal{F}_{\mathsf{vodm}}$ primitives specified in Figure 14.
- An OKVS scheme $(\mathsf{Encode}, \mathsf{Decode})$.
- A collision-resistant hash function $h(x) : \{0,1\}^* \to \{0,1\}^\sigma$.

Input of $\mathcal{S}$: $X = \{x_1, \ldots, x_n\}$
Input of $\mathcal{C}$: $Y = \{y_1, \ldots, y_n\}$
Protocol:

1. $\mathcal{S}$ selects a random *indication string* $s \in \mathbb{F}_{2^\sigma}$. $\mathcal{S}$ also runs $pp \leftarrow \mathsf{Setup}(1^\kappa)$ and $\mathsf{KeyGen}(pp)$ to obtain a key k. Then, $\mathcal{S}$ runs $\mathsf{Enc}_k(s)$ for n times to obtain $(s_1, \ldots, s_n)$.
2. $\mathcal{S}$ computes an OKVS $D := \mathsf{Encode}((h(x_1), s_1), \ldots, (h(x_n), s_n))$.
3. $\mathcal{S}$ sends D to $\mathcal{C}$.
4. $\mathcal{C}$ computes $s_i^* := \mathsf{Decode}(D, h(y_i))$ for $i \in [n]$.
5. $\mathcal{S}$ and $\mathcal{C}$ invoke the VODM functionality $\mathcal{F}_{\mathsf{vodm}}$. $\mathcal{C}$ acts as sender with input $S = \{s_1^*, \ldots, s_n^*\}$ and $\mathcal{S}$ acts as receiver with input k, s. As a result, $\mathcal{C}$ receives nothing and $\mathcal{S}$ receives $b \in \{0,1\}^n$.

Fig. 15: mq-RPMT protocol of [ZCL+23]

Security against Malicious Client. Since the only messages received from the client are the input of VODM, a malicious client may use a set of random ciphertexts instead of the ciphertexts computed from the OKVS. Note that the simulator cannot detect this attack, the ciphertexts in these two cases

[16] This leakage will not cause any harm to the construction of PSU. However, we need the standard mq-RPMT in our usecase.

are indistinguishable because of the IND-CPA security of the encryption scheme. Another problem is that the simulator cannot extract the adversary's inputs from the ciphertexts, which results in the simulator not being able to simulate correctly in the ideal world.

A.3 Mq-RPMT of Chen et al. [CZZ+24]

Chen et al. [CZZ+24] proposed a mq-RPMT based on a newly introduced cryptographic primitive called commutative weak pseudorandom function (cwPRF). Simply speaking, cwPRF is a family of PRF satisfying commutative property, that is, for any two keys k_1, k_2 and an input x, it satisfies $F_{k_1}(F_{k_2}(x)) = F_{k_2}(F_{k_1}(x))$. They use the DDH-based PRF $F_k(x) = H(x)^k$ as their cwPRF instantiation. The main idea is similar to the DDH-based PSI-cardinality protocol [CGT12]. We give the mq-RPMT protocol of [CZZ+24] in Figure 16.

Parameters:

- Two parties: server $\mathcal{S}$ and client $\mathcal{C}$.
- A cwPRF $F : K \times D \to D$
- A hash function $\mathsf{H} : \{0,1\}^\ell \to D$.

Input of $\mathcal{S}$: $X = \{x_1, \ldots, x_n\}$
Input of $\mathcal{C}$: $Y = \{y_1, \ldots, y_n\}$
Protocol:

1. $\mathcal{S}$ picks $k_1 \xleftarrow{R} K$, then sends $\{F_{k_1}(\mathsf{H}(x_i))\}_{i\in[n]}$ to $\mathcal{C}$.
2. $\mathcal{C}$ picks $k_2 \xleftarrow{R} K$, then computes and sends $\{F_{k_2}(\mathsf{H}(y_i)))\}_{i\in[n]}$ to $\mathcal{S}$. $\mathcal{C}$ also computes $\{F_{k_2}(F_{k_1}(\mathsf{H}(x_i)))\}_{i\in[n]}$, picks a random permutation $\pi \xleftarrow{R} [n]$, then sends $\Omega := \{F_{k_2}(F_{k_1}(\mathsf{H}(x_{\pi(i)})))\}_{i\in[n]}$ to $\mathcal{S}$. An alternative choice instead of explicit shuffle is inserting $\{F_{k_2}(F_{k_1}(\mathsf{H}(x_i)))\}_{i\in[n]}$ to a Bloom filter, then sends the resulting filter to $\mathcal{S}$.
3. $\mathcal{S}$ computes $\{F_{k_1}(F_{k_2}(\mathsf{H}(y_i)))\}_{i\in[n]}$, then sets $e_i = 1$ iff $F_{k_1}(F_{k_2}(\mathsf{H}(y_i))) \in \Omega$.

Fig. 16: mq-RPMT protocol of [CZZ+24]

Security against Malicious Client. The messages received from the client are $\{F_{k_2}(\mathsf{H}(y_i)))\}_{i\in[n]}$ and Ω in Step 2. A malicious client may replace the elements in Ω with some random values. Since F is a PRF, the simulator cannot distinguish these two cases. Another problem is extraction. Though the simulator could observe the queries of the client to the random oracle H, it cannot determine whether these queries are used to generate $\{F_{k_2}(\mathsf{H}(y_i)))\}_{i\in[n]}$ because the pseudorandomness of F. As a result, the simulator cannot simulate the adversary's view correctly in the ideal world.

A.4 Mq-RPMT of Tu et al. [TCLZ23]

Tu et al. [TCLZ23] proposed a PSU protocol in the unbalanced setting. In fact, their construction is also based on mq-RPMT. The core building blocks of their protocol are Fully Homomorphic Encryption (FHE) and a newly introduced functionality named permuted matrix Private EQuality Test (pm-PEQT). In pm-PEQT, a server holding a matrix $\mathbf{R}'_{\alpha\times m}$ and a matrix permutation $\pi = (\pi_c, \pi_r)$ interacts with a client holding a matrix $\mathbf{R}_{\alpha\times m}$. As a result, the client learns (only) the bit matrix $\mathbf{B}_{\alpha\times m}$ indicating that $b_{ij} = 1$ if $r_{\pi(ij)} = r'_{\pi(ij)}$ and $b_{ij} = 0$ otherwise, for $i \in [\alpha], j \in [m]$, while the server learns nothing about $\mathbf{R}$. The formal definition of pm-PEQT functionality is given in Figure 17. Their main idea is to let the client use FHE to encrypt its element y and send the ciphertext $c = \mathsf{Enc}_{pk}(y)$ to the server. The server with input X homomorphically computing the cihpertext $c' = \mathsf{Enc}_{pk}(f(y))$, where $f(x) = \Pi_{x_i\in X}(x - x_i) + r$ and r is a random value. Then the server sends the ciphertext c' to the client. The client decrypts the ciphertext to obtain r'. Then the parties invoke a PEQT functionality to test whether $r = r'$. It is simple to check that if $y \in X, r' = f(y) = r$. To test n elements, they also use the hashing technique to reduce the cost. Since the mq-RPMT construction

Parameters: Sender $\mathcal{S}$, Receiver $\mathcal{R}$, matrix sizes α, m.
Functionality:

- Wait for an input $\mathbf{R}' = [r_{ij}]$, $i \in [\alpha], j \in [m]$ and a permutation $\pi = (\pi_c, \pi_r)$ from the sender $\mathcal{S}$.
- Wait for input $\mathbf{R} = [r_{ij}]$, $i \in [\alpha], j \in [m]$ from the receiver $\mathcal{R}$.
- Define a bit matrix $\mathbf{B}_{\alpha \times m} = [b_{ij}]$, where $b_{ij} = 1$, if $r_{\pi(ij)} = r'_{\pi(ij)}$ and $b_{ij} = 0$ otherwise, for $i \in [\alpha], j \in [m]$.
- Give output $b \in \{0,1\}^n$ to the receiver $\mathcal{R}$.

Fig. 17: Permuted Matrix Private Equality Test Functionality $\mathcal{F}_{\mathsf{pm\text{-}PEQT}}$

Parameters:

- Two parties: server $\mathcal{S}$ and client $\mathcal{C}$.
- Ideal $\mathcal{F}_{\mathsf{pm\text{-}PEQT}}$ primitives specified in Figure 17.
- An FHE scheme $\mathcal{E} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$.

Input of $\mathcal{S}$: $X = \{x_1, \ldots, x_n\}$
Input of $\mathcal{C}$: $Y = \{y_1, \ldots, y_n\}$
Protocol:

1. $\mathcal{C}$ hashes Y into table Y_c by Cuckoo hashing, where Y_c consists of m bins and each bin has only one item. $\mathcal{S}$ uses the same hash functions to hash X into table $\mathbf{X}_{B \times m}$ by simple hashing, where $\mathbf{X}_{B \times m}$ consists of m bins and each bin has B items.
2. $\mathcal{C}$ generates the FHE key (pk, sk) and computes the FHE ciphertexts $c_j := \mathsf{Enc}_{pk}(Y[j]), j \in [m]$. Then, $\mathcal{C}$ sends these ciphertexts $\{c_j\}_{j \in [m]}$ to the server.
3. $\mathcal{S}$ partitions $\mathbf{X}_{B \times m}$ by rows into α subtables $\mathbf{X}_1, \cdots, \mathbf{X}_\alpha$. Each subtable has $B' = B/\alpha$ rows and m columns. Then, $\mathcal{S}$ picks random values $r_{i,j}$ and defines the polynomial $f_{i,j}(x) := \Pi_{x_k \in X_i[j]}(x - x_k) + r_{i,j}, i \in [\alpha], j \in [m]$, where $X_i[j]$ denotes the j-th column of X_i.
4. $\mathcal{S}$ homomorphically computes ciphertexts $c_{i,j} = \mathsf{Enc}_{pk}(f_{i,j}(Y[j]))$ for $i \in [\alpha], j \in [m]$ and sends $\{c_{i,j}\}_{i \in [\alpha], j \in [m]}$ to $\mathcal{C}$.
5. $\mathcal{C}$ decrypts $r'_{i,j} := \mathsf{Dec}_{sk}(c_{i,j})$ for $i \in [\alpha], j \in [m]$ and picks a random permutation π.
6. $\mathcal{S}$ and $\mathcal{C}$ invoke the pm-PEQT functionality $\mathcal{F}_{\mathsf{pm\text{-}PEQT}}$. $\mathcal{S}$ acts as receiver with input $R_{i \in [\alpha], j \in [m]} = \{r_{i,j}\}$ and $\mathcal{C}$ acts as sender with input $R'_{i \in [\alpha], j \in [m]} = \{r'_{i,j}\}$ and π. As a result, $\mathcal{C}$ receives nothing and $\mathcal{S}$ receives $B_{i \in [\alpha], j \in [m]} = \{b_{i,j}\} \in \{0,1\}^{\alpha \times m}$, where $b_{i,j} = 1$ if and only if $r_{\pi(ij)} = r'_{\pi(ij)}$.
7. $\mathcal{S}$ defines a bit vector $\mathbf{b} = [b_j]$, $j \in [m]$, where $b_j = 1$ if for all $i \in [\alpha]$, $b_{i,j} = 0$ and $b_j = 0$ otherwise.

Fig. 18: mq-RPMT protocol of [TCLZ23]

of Tu et al. [TCLZ23] uses many FHE optimization techniques, we just provide a simplified version to demonstrate their ideas more clearly. The formal description is in Figure 18.

Security against Malicious Client. To achieve security against malicious client, a base requirement is to replace all the sub-protocols with the malicious secure one. However, it seems difficult to enhance the security of pm-PEQT protocol in [TCLZ23]. Also, the simulator cannot extract the inputs of a malicious client in step 2 because of the security of the FHE scheme. As a result, the simulator cannot simulate the adversary's view correctly in the ideal world.

B Oblivious Switching Network

We give the definition of switching network as follows.

Definition 6 (Switching Network). *A switching network is a circuit (dag) with the following kinds of gates. Each gate has primary inputs/outputs as well as a programming input.*

- *A **multiplexer** takes k primary inputs and selects one of them to transfer to its single primary output. The choice of input is determined by the programming input (an element of $[k]$).*
- *A **permute switch** maps its two primary inputs to its primary outputs using a permutation selected by the programming input (a single bit).*

If $\mathbb{S}$ is a switching network, then we write $\mathbb{S}^p(a_1, \ldots, a_{n_{in}}) = (b_1, \ldots, b_{n_{out}})$ to denote that on programming inputs p, and primary inputs $a_1, \ldots, a_{n_{in}}$, the network outputs $b_1, \ldots, b_{n_{out}}$. In this paper, we only consider the permutation network, that is, $n_{in} = n_{out} = n$.

Parameters:

– Two parties: sender $\mathcal{S}$ and receiver $\mathcal{R}$. A switching network $\mathbb{S}$ with n inputs/outputs.

Input of $\mathcal{S}$: $X = \{x_1, \ldots, x_n\} \subset \{0,1\}^l$
Input of $\mathcal{R}$: A programming sequence π for $\mathbb{S}$
Protocol:

1. For every wire i in $\mathbb{S}$, the sender $\mathcal{S}$ chooses random value $M_i \leftarrow \{0,1\}^l$.
2. For every gate g in $\mathbb{S}$, $\mathcal{S}$ and $\mathcal{R}$ invoke an oblivious transfer protocol, where the receiver's input is the programming input π_g for that gate.
   (a) If g is a k-multiplexer with input wires $i_1, \ldots, i_k$ and output wire j, then the OT is a 1-out-of-k OT and the sender's input is $(M_{i_1} \oplus M_j, \ldots, M_{i_k} \oplus M_j)$.
   (b) If g is a permute switch with input wires i_1, i_2 and output wires j_1, j_2, then the OT is a 1-out-of-2 OT and the sender's input is $((M_{i_1} \oplus M_{j_1})||(M_{i_2} \oplus M_{j_2}), (M_{i_2} \oplus M_{j_1})||(M_{i_1} \oplus M_{j_2}))$.
3. For every input wire i, the sender sends $x_i \oplus M_i$ to the receiver.
4. For every output wire w, the receiver identifies the path from the output wire to the corresponding input wire $\pi(w)$. Say that path contains wires $i_1, \ldots, i_k$. Then from step 2 the receiver has $M_{i_j} \oplus M_{i_{j+1}}$ for every j, and from step 3 the receiver has $M_{i_1} \oplus x_{\pi(w)}$. XORing all of these together, the receiver obtains $x_{\pi(w)} \oplus M_w$ where M_w is the mask on the output wire w.
5. The sender outputs the collection of output wire masks. The receiver outputs the collection of $x_{\pi(w)} \oplus M_w$ values computed in the previous step.

Fig. 19: Oblivious Switching Network Protocol of [MS13]

Theorem 3. *The protocol in Figure 19 realizes the OSN functionality $\mathcal{F}_{\mathsf{osn}}$ against a malicious receiver in the $\mathcal{F}_{\mathsf{ot}}$-hybrid model.*

Proof. For the corrupt *malicious* receiver, we exhibit the simulator $\mathsf{Sim}_{\mathcal{R}}$ as follows:

1. In step 2, for each gate g, the simulator receives the programming input π_g from the adversary and sends a random string M_g to the receiver.
2. The simulator defines a programming sequence π from π_g received from the adversary, and input π to the ideal functionality. As a result, the simulator receives $x_{\pi(w)} \oplus M_w$ for each output wire w.
3. For every output wire i, the simulator identifies the path from the input wire to the corresponding output wire w. Say that path contains wires $i_1, \ldots, i_k$ and the corresponding values returned from the simulator are $M_{i_1}, \ldots, M_{i_k}$. The simulator sends $x_{\pi(w)} \oplus M_w \bigoplus_{j \in [k]} M_{i_j}$ to the adversary.

The correctness of this simulator directly follows the security of OT protocols.

C Payable PSI from Permuted mq-PMT of Cristofaro et al. [CGT12]

In this section, we give another payable PSI construction. The main idea comes from the permuted mq-PMT of Cristofaro et al. [CGT12], which dates back to the DH-PSI. We first review the DH-PSI as follows. Simply speaking, for a hash function H modeled as a random oracle and underlying acyclic group $\mathbb{G}$ of order q, for which the DDH problem is hard, the server and the client sample α and β, respectively, from $\mathbb{Z}_q$. The server computes $\bar{X} = \{H(x_i)^\alpha\}_{i \in [n]}$ and sends $\bar{X}$ to the client. The client computes $\bar{Y} = \{H(y_i)^\beta\}_{i \in [n]}$ and sends $\bar{Y}$ to the server. The server, then, computes $Y' = \{(H(y_i)^\beta)^\alpha\}_{i \in [n]}$ and sends it to the client. The client computes $X' := \{(H(x_i)^\alpha)^\beta\}_{i \in [n]}$ and output $\{y_i : H(y_i)^{\beta\alpha} \in X'\}$. Note that the above protocol could also allow the server to obtain an intersection, having the client send X' back to the server.

To let the server only obtain cardinality, the main idea is simply to have the client permute X' under a random permutation π before sending it to the server. In this way, the server obtains the result of permuted mq-PMT, that is, for the i-th item x'_i in X', we have $x'_i \in Y' \iff x_{\pi(i)} \in Y$.

However, the above protocol does not secure against malicious client. A malicious client can make the server output 0 with overwhelming probability by sending a random X'. We should enable the server to verify that the set X' sent by the client is correct. Our idea is to let the client send X' along with a zero knowledge proof that X' is honestly generated. However, as pointed by Miao et al. [MPR+20], it is challenging to prove the knowledge of a pre-image for a hash value. To address this problem, we adapt the malicious PSI-sum protocol of [MPR+20] to our setting. The main observation is that the DH-PSI protocol can be viewed as a distributed OPRF under PRF instantiation $F_k(x) = H(x)^k$ [BR93]. By replacing it with the Dodis-Yampolskiy PRF $F_k(x) = g^{\frac{1}{x+k}}$ [DY05], we can use sigma protocols and the shuffle-and-decrypt protocol of [BG12] to provide zero-knowledge proofs for the correctness of the PRF and the permutation, respectively.

We give the new payable PSI protocol in Figure 20 and 21. For simplicity, we refer to [MPR+20] for more details about Pedersen commitment [Ped91], Camenisch-Shoup encryption [CS03], ElGamal encryption [Gam84], and zero knowledge proofs [BG12] used in this protocol.

The advantage of this protocol is that it can easily be transformed into a protocol that is secure against malicious server, and the communication is linear $O(n)$, where n is the size of the set. However, the efficiency of this protocol is several magnitudes slower than the payable PSI protocol in Section 5.

Theorem 4. *The protocol in Figure 20 and 21 along with the same algorithm* $(\mathsf{Check}, \mathsf{Pricing})$ *as in Section 5 is a payable implement of* $\mathcal{F}_{\mathsf{psi}}$.

Proof. For the corrupt *simi-honest* server, we exhibit the simulator $\mathsf{Sim}_{\mathcal{S}}(X, judge)$ as follows.

1. In the offline stage, the simulator $\mathsf{Sim}_{\mathcal{S}}$ executes as an honest client $\mathcal{C}$ to generate the parameters and appends $(\mathsf{pk}_2^{cs}, \mathsf{pk}_2^{eg})$ and zero knowledge proofs to the view.
2. In step 4, the simulator commits to 0 instead of y_i and replace corresponding $\mathsf{ZK\text{-}AOK}$ with a simulated one.
3. In steps 5 and 6, the simulator behaves as an honest client to obtain g and appends $(\mathsf{ct}_{k_2}, \mathsf{C}_{k_2})$ and corresponding $\mathsf{ZK\text{-}AOK}$ to the view.
4. In steps 8 and 9, the simulator behaves as an honest client to obtain $\{\sigma_{1,i}\}_{i\in[n_1]}$ and appends $\{(\mathsf{C}_{\beta_{1,i}}, \mathsf{ct}_{\sigma_{1,i}}, \mathsf{ct}_{\sigma_{1,\pi(i)}})\}_{i\in[n_1]}$ and corresponding $\mathsf{ZK\text{-}AOK}$ to the view.
5. In step 12, the simulator replaces all the commitments of $a_{2,i}, b_{2,i}, \alpha_{2,i}$ with the commitments of 0, that is, $\mathsf{C}_{a_{2,i}} \leftarrow \mathsf{com}_{g_1^p,h_1^p}(0), \mathsf{C}_{b_{2,i}} \leftarrow \mathsf{com}_{g_1^p,h_1^p}(0), \mathsf{C}_{\alpha_{2,i}} \leftarrow \mathsf{com}_{g_1^p,h_1^p}(0)$. Then, the simulator selects $\sigma_{2,i}$ randomly conditioned on $|\{\sigma_{2,i}\} \cap \{\sigma_{1,i}\}| = judge$. The simulator picks random $\gamma_{2,i} \leftarrow [q^2 \cdot 2^\lambda]$ and computes $g_{2,i} = \sigma_{2,i}^{\gamma_{2,i}^{-1}}, \mathsf{ct}_{\beta_{2,i}} = \mathsf{CS.Enc}_{\mathsf{pk}_1^{cs}}(\gamma_{2,i}^{-1})$. As a result, the simulator appends $\{(\mathsf{C}_{a_{2,i}}, \mathsf{C}_{b_{2,i}}, \mathsf{C}_{\alpha_{2,i}}, \mathsf{ct}_{\beta_{2,i}}, g_{2,i})\}_{i\in[n_2]}$ and a simulated $\mathsf{ZK\text{-}AOK}$ to the view.

We show the correctness of this simulation via a sequence of hybrids:

- Hybrid_0. The first hybrid is the real interaction described in Figure 20 and 21. Here, the honest client uses input Y, and honestly interacts with the corrupt server. Let T_0 denote the real view of the server.
- Hybrid_1. Let T_1 be the same as T_0, except that the commitments in step 12 are replaced with the commitments of 0 and the zero knowledge proofs are also replaced with a simulated one. This hybrid is computationally indistinguishable from T_0 by the hiding property of the commitment scheme and the zero knowledge property of $\mathsf{ZK\text{-}AOK}$.
- Hybrid_2. Let T_2 be the same as T_1, except that the order of computation is changed as follows: the client computes $\sigma_{2,i} = F_{k_1+k_2}(y_i)$, selects random $\gamma_{2,i} \leftarrow [q^2 \cdot 2^\kappa]$, and computes $g_{2,i} = \sigma_{2,i}^{\gamma_{2,i}^{-1}}, \mathsf{ct}_{\beta_{2,i}} = \mathsf{CS.Enc}_{\mathsf{pk}_1^{cs}}(\gamma_{2,i}^{-1})$. The client also updates the corresponding $\mathsf{ZK\text{-}AOK}$.
  Note that $\beta_{2,i}$ is randomly selected in the previous hybrid and $\gamma_{2,i}$ is decided by $\beta_{2,i}$. The new computation order keeps the distribution unchanged. By the zero knowledge property of $\mathsf{ZK\text{-}AOK}$, T_2 and T_1 are computationally indistinguishable.
- Hybrid_3. Let T_3 be the same as T_2, except that ct_{k_2} in step 6 is replaced with an encryption of 0 and the client updates the corresponding $\mathsf{ZK\text{-}AOK}$.
  By the semantic security of the CS encryption scheme and the zero knowledge property of $\mathsf{ZK\text{-}AOK}$, T_3 and T_2 are computationally indistinguishable.

Parameters:

- Two parties: server $\mathcal{S}$ and client $\mathcal{C}$.
- A group $\mathbb{G}$ of order q with a generator $\tilde{g}$ for which the $\max(n_1, n_2)$-DHI assumption holds.

Input of $\mathcal{S}$: $X = \{x_1, \ldots, x_{n_1}\}$
Input of $\mathcal{C}$: $Y = \{y_1, \ldots, y_{n_2}\}$
Protocol:
Offline Stage:

1. The server $\mathcal{S}$ generates parameters for encryption and commitment schemes. That is, $\mathcal{S}$ generates:
   (a) Key pairs of Camenisch-Shoup encryption scheme: $(\mathsf{pk}_1^{cs}, \mathsf{sk}_1^{cs}) \leftarrow \mathsf{CS.KeyGen}(1^\kappa)$, where $g_1 = r_1^{2N_1}$ for a random $r_1 \in \mathbb{Z}_{N_1^2}$, $\mathsf{pk}_1^{cs} = (N_1, g_1, y_1), N_1 \geq 2^{3\kappa}q^2, y_1 = g_1^{x_1}$ and $\mathsf{sk}_1^{cs} = x_1$
   (b) Pedersen commitment parameters: $(g_1^p, h_1^p) \leftarrow \mathsf{Per.Gen}(1^\kappa)$ where g_1^p, h_1^p are from the large subgroup of $\mathbb{Z}_{N_1}^*$.
   (c) Key pairs of ElGamal encryption scheme: $(\mathsf{pk}_1^{eg}, \mathsf{sk}_1^{eg}) \leftarrow \mathsf{EG.KeyGen}(1^\kappa)$, where $\mathsf{pk}_1^{eg} = \tilde{g}^{\mathsf{sk}_1^{eg}} \in \mathbb{G}$ and $\mathsf{sk}_1^{eg} \in \mathbb{Z}_q^*$.
   After generating these parameters, $\mathcal{S}$ sends $(\mathsf{pk}_1^{cs}, g_1^p, h_1^p, \mathsf{pk}_1^{eg})$ to the client $\mathcal{C}$.
2. The client $\mathcal{C}$ generates parameters along with the zero knowledge proofs for encryption and commitment schemes. That is, $\mathcal{C}$ generates:
   (a) Key pairs of Camenisch-Shoup encryption scheme: $(\mathsf{pk}_2^{cs}, \mathsf{sk}_2^{cs}) \leftarrow \mathsf{CS.KeyGen}(1^\kappa)$, where $g_2 = r_2^{2N_2}$ for a random $r_2 \in \mathbb{Z}_{N_2^2}$, $\mathsf{pk}_2^{cs} = (N_2, g_2, y_2), N_2 \geq 2^{3\kappa}q^2, y_2 = g_2^{x_2}$ and $\mathsf{sk}_2^{cs} = x_2$. A zero knowledge proof that N_2 is a product of two large safe primes and that y_2 is correctly formed: $\mathsf{ZK}\{x_2 : y_2 = (g_2)^{x_2} \bmod N_2^2\}$.
   (b) Key pairs of ElGamal encryption scheme: $(\mathsf{pk}_2^{eg}, \mathsf{sk}_2^{eg}) \leftarrow \mathsf{EG.KeyGen}(1^\kappa)$, where $\mathsf{pk}_2^{eg} = \tilde{g}^{\mathsf{sk}_2^{eg}} \in \mathbb{G}$ and $\mathsf{sk}_2^{eg} \in \mathbb{Z}_q^*$. A zero knowledge proof that $\mathsf{pk}_2^{eg} \in \langle\tilde{g}\rangle$: $\mathsf{ZK\text{-}AOK}\{\mathsf{sk}_2^{eg} : \mathsf{pk}_2^{eg} = (\tilde{g})^{\mathsf{sk}_2^{eg}}\}$.
   After generating these parameters, $\mathcal{C}$ sends $(\mathsf{pk}_2^{cs}, \mathsf{pk}_2^{eg})$ and the corresponding zero knowledge proofs to the server $\mathcal{S}$.
3. $\mathcal{S}$ and $\mathcal{C}$ generate a PRF key $k_1, k_2 \in \mathbb{Z}_q^*$, respectively.

Online Stage:

4. $\mathcal{C}$ computes the Pedersen commitments $\{\mathsf{C}_{y_i}\}_{i\in[n_2]}$ along with a zero knowledge proof $\mathsf{ZK\text{-}AOK}\{\{y_i, s_i\}_{i\in[n_2]} : \mathsf{C}_{y_i} = (g_1^p)^{y_i} \cdot (h_1^p)^{s_i}\}$ and sends them to $\mathcal{S}$.
5. $\mathcal{S}$ and $\mathcal{C}$ jointly decide on a random generator g for the group $\mathbb{G}$.
6. The client $\mathcal{C}$ computes $\mathsf{ct}_{k_2} \leftarrow \mathsf{CS.Enc}_{\mathsf{pk}_2^{cs}}(k_2)$ and $\mathsf{C}_{k_2} \leftarrow \mathsf{com}_{g_1^p, h_1^p}(k_2)$. Let $\mathsf{pk}_2^{cs} = (N_2, g_2, y_2), \mathsf{ct}_{k_2} = (u, e)$, $\mathcal{C}$ sends ct_{k_2} and C_{k_2} to $\mathcal{S}$ along with a zero knowledge proof $\mathsf{ZK\text{-}AOK}\{(k_2, r, r') : u = g_2^r \wedge e = (1 + N_2)^{k_2} \cdot y_2^r \wedge \mathsf{C}_{k_2} = (g_1^p)^{k_2} \cdot (h_1^p)^{r'} \wedge k_2 \leq q \cdot 2^{2\kappa+1}\}$.
7. The server $\mathcal{S}$ verifies the $\mathsf{ZK\text{-}AOK}$ received from $\mathcal{C}$, and abort if verification fails. Then, for $i \in [n_1]$, $\mathcal{S}$ computes the following:
   (a) Pick a random $a_{1,i} \leftarrow \mathbb{Z}_q^*$ and $b_{1,i} \leftarrow [q \cdot 2^\kappa]$. Compute $g_{1,i} = g^{a_{1,i}}$.
   (b) Compute $\alpha_{1,i} = a_{1,i} \cdot (k_1 + x_i)$.
   (c) Compute $\mathsf{ct}_{\beta_{1,i}} = (\mathsf{ct}_{k_2})^{a_{1,i}} \cdot \mathsf{CS.Enc}_{\mathsf{pk}_2^{cs}}(\alpha_{1,i}) \cdot (\mathsf{CS.Enc}_{\mathsf{pk}_2^{cs}}(b_{1,i}))^q$. We note here $\beta_{1,i} = a_{1,i} \cdot (k_1 + k_2 + x_i) + b_{1,i} \cdot q$.
   The server $\mathcal{S}$ sends $\{(\mathsf{ct}_{\beta_{1,i}}, g_{1,i})\}_{i\in[n_1]}$ to $\mathcal{C}$.

Fig. 20: Payable Private Set Intersection Protocol $\Pi_{\mathsf{p\text{-}psi}}$

8. For $i \in [n_1]$, the client $\mathcal{C}$ computes the following:
   (a) Compute $\beta_{1,i} = \mathsf{CS.Dec}_{\mathsf{sk}_2^{cs}}(\mathsf{ct}_{\beta_{1,i}})$ and $\mathsf{C}_{\beta_{1,i}} \leftarrow \mathsf{com}_{g_1^p,h_1^p}(\beta_{1,i})$.
   (b) Compute $\gamma_{1,i} = \beta_{1,i}^{-1} \bmod q$ and $\sigma_{1,i} = g_{1,i}^{\gamma_{1,i}}$. Check all the $\sigma_{1,i}$'s are distinct and abort otherwise.
   (c) Compute $\mathsf{ct}_{\sigma_{1,i}} = \mathsf{EG.Enc}_{\mathsf{pk}_2^{eg}}(\sigma_{1,i})$.
   (d) Generate a zero knowledge proof that $(\mathsf{C}_{\beta_{1,i}}, \mathsf{ct}_{\sigma_{1,i}})$ are honestly computed:

$$\begin{aligned}\mathsf{ZK\text{-}AOK}\{(\mathsf{sk}_2^{cs}, \beta_{1,i}, r_1, r_2) : & \beta_{1,i} = \mathsf{CS.Dec}_{\mathsf{sk}_2^{cs}}(\mathsf{ct}_{\beta_{1,i}}) \wedge \\ & \mathsf{C}_{\beta_{1,i}} = (g_1^p)^{\beta_{1,i}} \cdot (h_1^p)^{r_1} \wedge \beta_{1,i} \leq q^2 \cdot 2^{3\lambda+1} \wedge \\ & \mathsf{ct}_{\sigma_{1,i}} = \mathsf{EG.Enc}_{\mathsf{pk}_2^{eg}}((g_{1,i})^{\beta_{1,i}^{-1}}; r_2)\}\end{aligned}$$

   (e) Re-randomize $\mathsf{ct}_{\sigma_{1,i}}$ to $\mathsf{ct}'_{\sigma_{1,i}}$ with randomness 0

   $\mathcal{C}$ selects a random permutation π over $[n_1]$ and generates a zero knowledge proof that the ciphertexts are permuted correctly: $\mathsf{ZK\text{-}AOK}\{(\pi, \{r_i\}_{i\in[n_1]}) : \mathsf{ct}'_{\sigma_{1,\pi(i)}} = \mathsf{ct}_{\sigma_{1,i}} \cdot \mathsf{EG.Enc}_{\mathsf{pk}_2^{eg}}(1; r_i), i \in [n_1]\}$.
9. $\mathcal{C}$ sends $\{(\mathsf{C}_{\beta_{1,i}}, \mathsf{ct}_{\sigma_{1,i}}, \mathsf{ct}'_{\sigma_{1,\pi(i)}})\}_{i\in[n_1]}$ and all the $\mathsf{ZK\text{-}AOK}$ generated above to the server $\mathcal{S}$.
10. The server $\mathcal{S}$ verifies the $\mathsf{ZK\text{-}AOK}$ received from $\mathcal{C}$, and abort if verification fails. Note that the ciphertexts $\{\mathsf{ct}'_{\sigma_{1,\pi(i)}}\}$ have randomness 0, the server $\mathcal{S}$ obtains $F_k(X) = \{\sigma_{1,\pi(i)}\}_{i\in[n_1]}$.
11. The server $\mathcal{S}$ computes $\mathsf{ct}_{k_1} \leftarrow \mathsf{CS.Enc}_{\mathsf{pk}_1^{cs}}(k_1)$ and sends it to $\mathcal{C}$.
12. For $i \in [n_2]$, the client $\mathcal{C}$ computes the following:
   (a) Pick a random $a_{2,i} \leftarrow \mathbb{Z}_q^*$ and $b_{2,i} \leftarrow [q \cdot 2^\kappa]$. Compute $g_{2,i} = g^{a_{2,i}}$.
   (b) Compute $\alpha_{2,i} = a_{2,i} \cdot (k_2 + y_i)$ and commitments $\mathsf{C}_{a_{2,i}} \leftarrow \mathsf{com}_{g_1^p,h_1^p}(a_{2,i}), \mathsf{C}_{b_{2,i}} \leftarrow \mathsf{com}_{g_1^p,h_1^p}(b_{2,i}), \mathsf{C}_{\alpha_{2,i}} \leftarrow \mathsf{com}_{g_1^p,h_1^p}(\alpha_{2,i})$.
   (c) Compute $\mathsf{ct}_{\beta_{2,i}} = (\mathsf{ct}_{k_1})^{a_{2,i}} \cdot \mathsf{CS.Enc}_{\mathsf{pk}_1^{cs}}(\alpha_{2,i}) \cdot (\mathsf{CS.Enc}_{\mathsf{pk}_1^{cs}}(b_{2,i}))^q$. We note here $\beta_{2,i} = a_{2,i} \cdot (k_1 + k_2 + y_i) + b_{2,i} \cdot q$.

   The client $\mathcal{C}$ sends $\{(\mathsf{C}_{a_{2,i}}, \mathsf{C}_{b_{2,i}}, \mathsf{C}_{\alpha_{2,i}}, \mathsf{ct}_{\beta_{2,i}}, g_{2,i})\}_{i\in[n_2]}$ to $\mathcal{S}$ along with the zero knowledge proof :

$$\begin{aligned}\mathsf{ZK\text{-}AOK}\{(a_{2,i}, b_{2,i}, \alpha_{2,i}, r_1, r_2, r_3, r_4, r_5, r_6) : & \\ & \mathsf{C}_{a_{2,i}} = (g_1^p)^{a_{2,i}} \cdot (h_1^p)^{r_1} \wedge a_{2,i} \leq q \cdot 2^{2\lambda+1} \wedge \\ & \mathsf{C}_{b_{2,i}} = (g_1^p)^{b_{2,i}} \cdot (h_1^p)^{r_2} \wedge b_{2,i} \leq q \cdot 2^{3\lambda+1} \wedge \\ & \mathsf{C}_{\alpha_{2,i}} = (g_1^p)^{\alpha_{2,i}} \cdot (h_1^p)^{r_3} \wedge \mathsf{C}_{\alpha_{2,i}} = (\mathsf{C}_{k_2} \cdot \mathsf{C}_{y_i})^{a_{2,i}} \cdot (h_1^p)^{r_4} \wedge \alpha_{2,i} \leq q \cdot 2^{2\lambda+1} \wedge \\ & \mathsf{ct}_{\beta_{2,i}} = (\mathsf{ct}_{k_1})^{a_{2,i}} \cdot \mathsf{CS.Enc}_{\mathsf{pk}_1^{cs}}(\alpha_{2,i}; r_5) \cdot (\mathsf{CS.Enc}_{\mathsf{pk}_1^{cs}}(b_{2,i}; r_6))^q \wedge g_{2,i} = g^{a_{2,i}}\}\end{aligned}$$

13. For $i \in [n_2]$, the server $\mathcal{S}$ computes the following:
   (a) Compute $\beta_{2,i} = \mathsf{CS.Dec}_{\mathsf{sk}_1^{cs}}(\mathsf{ct}_{\beta_{2,i}})$.
   (b) Compute $\gamma_{2,i} = \beta_{2,i}^{-1} \bmod q$ and $\sigma_{2,i} = g_{2,i}^{\gamma_{2,i}}$. Check all the $\sigma_{2,i}$'s are distinct and abort otherwise.
   (c) Compute $\mathsf{ct}_{\sigma_{2,i}} = \mathsf{EG.Enc}_{\mathsf{pk}_1^{eg}}(\sigma_{2,i})$.
14. $\mathcal{S}$ sends $F_k(Y) = \{\sigma_{2,i}\}_{i\in[n_2]}$ to the client $\mathcal{C}$. Then, the server $\mathcal{S}$ defines and outputs $judge := |X \cap Y| = |\{t : \sigma_{1,t} \in F_k(Y)\}|$
15. The client $\mathcal{C}$ defines and outputs $X \cap Y := \{y_t : \sigma_{2,t} \in F_k(X)\}$.

Fig. 21: Payable Private Set Intersection Protocol $\Pi_{\mathsf{p\text{-}psi}}$, continued

– Hybrid$_4$. Let T_4 be the same as T_3, except that $\sigma_{2,i}$'s are selected randomly conditioned on $|\{\sigma_{2,i}\} \cap \{\sigma_{1,i}\}| = judge$. By the pseudorandomness PRF, T_4 and T_3 are computationally indistinguishable.
– Hybrid$_5$. Let T_5 be the same as T_4, except that ct_{k_2} in step 6 is changed again with the real encryption of k_2 and the client updates the corresponding ZK-AOK.
By the semantic security of the CS encryption scheme and the zero knowledge property of ZK-AOK, T_5 and T_4 are computationally indistinguishable.
– Hybrid$_6$. Let T_6 be the same as T_5, except that the commitments in step 4 are replaced with the commitments of 0 and the zero knowledge proofs are also replaced with a simulated one. The hiding property of the commitment scheme and the zero knowledge property of ZK-AOK guarantee that T_6 and T_5 are computationally indistinguishable. This hybrid is exactly the view output by the simulator.

For the corrupt *malicious* client, we exhibit the simulator $\mathsf{Sim}_{\mathcal{R}}$ as follows:

1. In the offline stage, the simulator $\mathsf{Sim}_{\mathcal{R}}$ executes as an honest server $\mathcal{S}$ to generate the parameters and sends $(\mathsf{pk}_1^{cs}, \mathsf{pk}_1^{eg}, g_1^p, h_1^p)$ to the adversary. We note that in the following step, the simulator sends abort to the functionality $\mathcal{F}_{\mathsf{psi}}$ if any zero knowledge proof received from the adversary does not verify.
2. In step 4, the simulator extracts the adversary's input $Y = \{y_i\}_{i\in[n_2]}$ from the ZK-AOK. Then, the simulator sends Y to the functionality $\mathcal{F}_{\mathsf{psi}}$ and obtains the intersection $I = X \cap Y$.
3. In step 5, the simulator behaves as an honest server to obtain g.
4. In step 7, the simulator picks random $\sigma_{1,i} \leftarrow \mathbb{G}, \gamma_{1,i} \leftarrow [q^2 \cdot 2^\lambda]$, computes $g_{1,i} = \sigma_{1,i}^{\gamma_{1,i}^{-1}}, \mathsf{ct}_{\beta_{1,i}} = \mathsf{CS.Enc}_{\mathsf{pk}_2^{cs}}(\gamma_{1,i}^{-1})$ and sends $\{\mathsf{ct}_{\beta_{1,i}}, g_{1,i}\}_{i\in[n_1]}$ to the adversary.
5. In steps 8-12, the simulator behaves as an honest server and sends abort to the functionality $\mathcal{F}_{\mathsf{psi}}$ if any verification fails.
6. In step 14, the simulator defines $\sigma_{2,i} = \sigma_{1,i}$ if $y_i \in I$ and picks random $\sigma_{2,i}$ if $y_i \notin I$ for $i \in [n_2]$. The simulator sends $\{\sigma_{2,i}\}_{i\in[n_2]}$ to the adversary.

We show the correctness of this simulation via a sequence of hybrids:

– Hybrid$_0$. The first hybrid is the real interaction described in Figure 20 and 21. Here, the honest server uses input X, and honestly interacts with the corrupt client. Let T_0 denote the real view of the client.
– Hybrid$_1$. Let T_1 be the same as T_0, except that the order of computation is changed as follows: in step 7, the server computes $\sigma_{1,i} = F_{k_1+k_2}(x_i)$, selects random $\gamma_{1,i} \leftarrow [q^2 \cdot 2^{\lambda-2}]$, and computes $g_{1,i} = \sigma_{1,i}^{\gamma_{1,i}^{-1}}, \mathsf{ct}_{\beta_{1,i}} = \mathsf{CS.Enc}_{\mathsf{pk}_2^{cs}}(\gamma_{1,i}^{-1})$.
Note that $\beta_{1,i}$ is randomly selected in the previous hybrid and $\gamma_{1,i}$ is decided by $\beta_{1,i}$. The new computation order keeps the distribution unchanged. As a result, T_1 and T_0 are statistically indistinguishable.
– Hybrid$_2$. Let T_2 be the same as T_1, except that ct_{k_1} in step 11 is replaced with an encryption of 0. By the semantic security of the CS encryption scheme, T_2 and T_1 are computationally indistinguishable.
– Hybrid$_3$. Let T_3 be the same as T_2, except that $\sigma_{2,i}$'s are selected randomly conditioned on $\sigma_{2,i} = \sigma_{1,i}$ if $y_i \in I$. By the pseudorandomness PRF, T_3 and T_2 are computationally indistinguishable.
– Hybrid$_4$. Let T_4 be the same as T_3, except that ct_{k_1} in step 11 is changed again with the real encryption of k_1.
By the semantic security of the CS encryption scheme, T_4 and T_3 are computationally indistinguishable. This hybrid is exactly the view output by the simulator.