Optimal Private Set Union from Multi-Query Reverse Private Membership Test

Cong Zhang[1,2], Yu Chen[3], Weiran Liu[4], Min Zhang[3], and Dongdai Lin[1,2]

[1] State Key Laboratory of Information Security, Institute of Information Engineering, Chinese Academy of Sciences
{zhangcong,ddlin}@iie.ac.cn,
[2] School of Cyber Security, University of Chinese Academy of Sciences
[3] School of Cyber Science and Technology, Shandong University
yuchen.prc@gmail.com, zm_min@mail.sdu.edu.cn
[4] Alibaba Group
weiran.lwr@alibaba-inc.com

Abstract. Private set union (PSU) protocol enables two parties, each holding a set, to compute the union of their sets without revealing anything else to either party. So far, there are two known approaches for constructing PSU protocols. The first mainly depends on additively homomorphic encryption (AHE), which is generally inefficient since it needs to perform a non-constant number of homomorphic computations on each item. The second is mainly based on oblivious transfer and symmetric-key operations, which is recently proposed by Kolesnikov et al. (KRTW, ASIACRYPT 2019). It features good practical performance, which is several orders of magnitude faster than the first one. However, neither of these two approaches is optimal in the sense that their computation and communication complexity are not both $O(n)$, where n is the size of the set. Therefore, the problem of constructing the optimal PSU protocol remains open.

In this work, we resolve this open problem by proposing a generic framework of PSU from oblivious transfer and a newly introduced protocol called multi-query reverse private membership test (mq-RPMT). We present two generic constructions of mq-RPMT. The first is based on symmetric-key encryption and general 2PC techniques. The second is based on re-randomizable public-key encryption. Both constructions lead to PSU with linear computation and communication complexity.

By instantiating the generic constructions of mq-RPMT, we obtain two concrete PSU protocols based on SKE and PKE techniques respectively. We implement our two PSU protocols and compare them with the state-of-the-art PSU. Experiments show that our PKE-based protocol has the lowest communication of all schemes, which is $4.1 - 14.8\times$ lower depending on set size. The running time of our PSU scheme is $1.2 - 12\times$ faster than that of state-of-the-art depending on network environments.

1 Introduction

Private set union (PSU) enables two parties, each holding a private set of elements, to compute the union of the two sets while revealing nothing more than the union itself. PSU and its variants have numerous applications, such as information security risk assessment [LV04], IP blacklist and vulnerability data aggregation [HLS+16], joint graph computation [BS05], distributed network monitoring [KS05], building block for private DB supporting full join [KRTW19], private ID [GMR+21] etc.

Over the last decade, there has been a significant amount of work on private set operation, especially private set intersection (PSI) [FNP04, PSZ14, KKRT16, PRTY19, CM20, PRTY20]. We refer the reader to [PSZ18] for an overview of different PSI paradigms. State-of-the-art semi-honest PSI protocols in the two-party setting [KKRT16, PRTY19, CM20, RS21, GPR+21] all mainly rely on symmetric-key operations, except for a few base oblivious transfer (OT) operations in OT extension protocol [IKNP03, KK13]. Let n denote the size of input set, the communication complexity of these OT-based PSI protocols has been improved from initial nonlinear $O(n \log n)$ [PSZ14, PSSZ15, KKRT16] to linear complexity $O(n)$ [PRTY19, FNO19, GN19, CM20, RS21, GPR+21].

1.1 Motivation

In contrast to the affairs of PSI, the efficiency of the state-of-the-art PSU is less satisfactory. Roughly, there are two known approaches for constructing PSU protocols. The first is mainly based on public-key techniques. Existing constructions along this approach [KS05, Fri07, HN10, SM18] have to perform a non-constant number of additively homomorphic encryption (AHE) operations on each set element, rendering the overall protocols inefficient. The other is mainly based on symmetric-key techniques in combination with OT [KRTW19, GMR+21], which is several orders of magnitude faster than AHE-based constructions. However, neither of the two approaches is optimal in the sense that their computation and communication complexity are not both $O(n)$, where n is the size of the set. We note that [DC17] is the work closest to optimal bound, but its communication and computation complexity additionally depend on the statistical security parameter λ. This leaves the following open problem:

Can we construct PSU protocols with linear computation and communication complexity?

1.2 Our Contribution

In this paper, we answer this question affirmatively in the semi-honest setting. Our contribution can be summarized as follows:

1. We revisit the state-of-the-art PSU protocol [KRTW19] (KRTW protocol for short hereafter) in depth. Roughly, KRTW protocol is built upon two building blocks, namely oblivious transfer (OT) and reverse private membership test (RPMT). We figure out the root causing KRTW protocol non-optimal is that RPMT has linear communication complexity and super-linear computation complexity, and it has to be carried out n times independently, where n is the size of sender's private set.
2. To achieve linear complexity, we propose a new framework for constructing PSU protocols. The core building block is a newly introduced protocol called multi-query RPMT (mq-RPMT). Compared with original RPMT [KRTW19], mq-RPMT enables a sender to test whether his n elements belong to receiver's input set X in a one-shot, rather than repeating n times.
   Multi-query RPMT dominates the complexity of the resulting PSU protocol. However, it is challenging to construct mq-RPMT. We identify and overcome several technical difficulties for building optimal mq-RPMT, and give a modular construction from oblivious key-value store (OKVS) [GPR+21] and a novel functionality called vector oblivious decryption-then-matching (VODM). We then give two realizations of mq-RPMT via carefully instantiating the OKVS and the VODM functionality. The first is mainly based on symmetric-key encryption (SKE) and 2PC techniques, while the second is mainly based on re-randomizable public-key encryption (ReRand-PKE). Both the two concrete mq-RPMT protocols achieve linear communication and computation complexity.
3. We further abstract a new primitive called membership encryption (ME), which broadens the scope of the candidate encryption scheme, unifies our two constructions, and halves the communication complexity of our SKE-based construction on receiver side.
4. Combining OT and the above mq-RPMT, we eventually obtain SKE-based and PKE-based PSU protocols with optimal complexity for the first time. Experiments show that our PKE-based protocol has the lowest communication of all schemes, which is $4.1-14.8\times$ lower depending on set size. The running time of our PSU scheme is $1.2-12\times$ faster than that of state-of-the-art depending on the network environment. In addition to our scheme, we also use Silent OT [BCG+19, YWL+20] to optimize the scheme of [GMR+21], and provide different parameter selection of Ferret OT [YWL+20].

Fig. 1 depicts the technical overview of our new PSU framework. We elaborate the details in the next subsection.

1.3 Overview of Our Techniques

We provide the high-level technical overview for our new framework of PSU protocol.

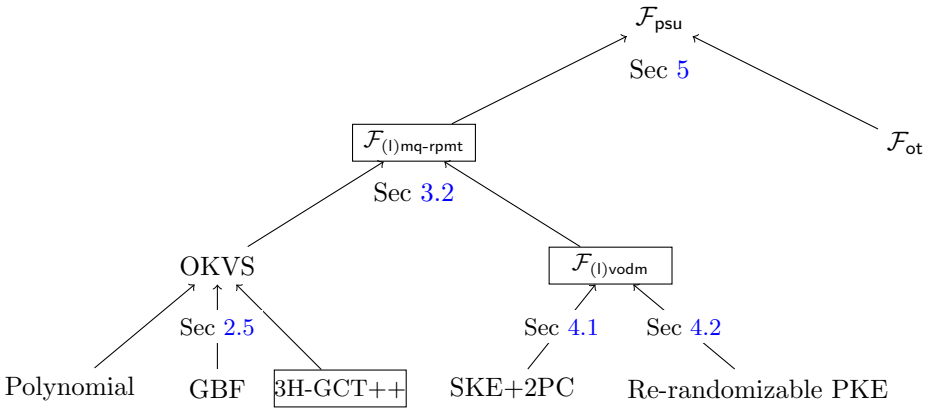

Fig. 1: Technical overview of our new PSU framework. The new primitives and functionalities are marked with rectangles.

KRTW protocol revisit. Our starting point is the recent PSU protocol of Kolesnikov et al. [KRTW19]. The core of KRTW protocol is a subprotocol called reverse private membership test (RPMT), which can test whether a sender's element y belongs to the receiver's input set X, and let the receiver obtain the result. After that, both parties execute OT protocol to let the receiver obtain $\{y\} \cup X$. The computation cost of original RPMT [KRTW19] is $O(n \log^2 n)$ and the communication cost is $O(n)$. For the purpose of computing the set union, the parties need to execute RPMT n times independently, which results in $O(n^2)$ communication and $O(n^2 \log^2 n)$ computation. The complexity can be further reduced to $O(n \log n)$ and $O(n \log n \log \log n)$ separately via hash to bin technology, but it is still *super-linear*. The bottleneck of the KRTW protocol is exactly RPMT.

Zoom in on the original RPMT. The original RPMT protocol employs an oblivious PRF (OPRF) functionality $\mathcal{F}_{\mathsf{oprf}}$ and a private equality test (PEQT) functionality $\mathcal{F}_{\mathsf{peqt}}$. In OPRF, the sender learns a random PRF key k and the receiver learns the PRF output $F_k(y_1), \ldots, F_k(y_n)$ on its inputs $y_1, \ldots, y_n \in Y$. In PEQT, the functionality receives two strings from the receiver and the sender respectively and tells the receiver whether the two strings are equal. Their RPMT uses an *indication string* s to indicate the membership of X.

More precisely, their RPMT protocol executes as follows with sender $\mathcal{S}$'s input y and receiver $\mathcal{R}$'s input $X = \{x_1, \ldots, x_n\}$: $\mathcal{S}$ and $\mathcal{R}$ execute the OPRF protocol first. The receiver $\mathcal{R}$ receives a PRF key k. The sender $\mathcal{S}$ inputs y, and receives $q^* = F_k(y)$. Next, $\mathcal{R}$ chooses a random indication string s. Then, $\mathcal{R}$ computes and sends the interpolation polynomial P which passes through points $\{(x_i, s \oplus F_k(x_i))\}_{i \in [n]}$ to the sender. After receiving P, $\mathcal{S}$ computes $s^* := q^* \oplus P(y)$. Now, $\mathcal{S}$ and $\mathcal{R}$ invoke the $\mathcal{F}_{\mathsf{peqt}}$-functionality with input s^* and s separately. Finally, $\mathcal{R}$ receives output from $\mathcal{F}_{\mathsf{peqt}}$.

If $y \in X$, i.e., there exists an x_i such that $y = x_i$, then we have $s^* = q^* \oplus P(y) = F_k(x_i) \oplus P(x_i) = s$. If $y \notin X$, then $q^* = F_k(y)$ is pseudorandom, which implies that $s^* = q^* \oplus P(y) \neq s$ with overwhelming probability.

To identify the root of the inefficiency of the original RPMT protocol, we first try to interpret it at an abstract level. Our first key observation is that the polynomial actually plays the role of oblivious key-value store (OKVS). Our second key observation is that the usage of OPRF is three-fold. Firstly, $\mathcal{R}$ uses an OPRF to derive n pseudorandom one-time pads, then encrypts the same indication string into n ciphertexts under these one-time pads. Secondly, $\mathcal{S}$ utilizes OPRF to decrypt a ciphertext obliviously. Finally, OPRF provides OKVS with randomness to ensure the correctness of the protocol.

Based on the above new interpretation, we are ready to describe our new mq-RPMT protocol in an incremental way over the original RPMT protocol.

Enhanced oblivious key-value store. One reason that accounts for the super-linear complexity of the original RPMT protocol is that the polynomial related operations are costly. More precisely, the complexity of polynomial interpolation is $O(n \log^2 n)$, and the amortized complexity of polynomial evaluation is $O(\log^2 n)$. According to our first observation, polynomial essentially plays the role of OKVS. This greatly increases the space of the concrete mapping schemes that can be used. A drop-in

replacement of polynomial with more efficient OKVS candidates can reduce the computation complexity immediately. However, as we observed before, an additional randomness property should be satisfied now, since we do not use OPRF to provide randomness anymore. To achieve this goal, we enhance OKVS in two aspects: efficiency and security. (See Section 2.5 for the details.)

Oblivious decryption-then-matching. Another reason that accounts for the super-linear complexity is that the original RPMT protocol is one-time in nature. To see this, note that in the original RPMT protocol $\mathcal{S}$ learns the purported indication string. This design lets $\mathcal{S}$ learn more information than needed, and is exactly the reason that hinders multi-query. For example, if there are two distinct elements belonging to $\mathcal{R}$'s set, then $\mathcal{S}$ will obtain the same indication string. This will let $\mathcal{S}$ know that the two elements belong to the intersection, which violates security.

Based on the above discussion, the rough idea of making RPMT support multi-query is to encode the *ciphertext* of indication string in OKVS instead of the indication string itself. In this way, $\mathcal{S}$ will obtain some ciphertexts (i.e. the value of $\mathsf{OKVS}(y)$), and $\mathcal{R}$ has the corresponding key. We need to let $\mathcal{R}$ decrypt these ciphertexts, and match the results with the indication string. A naive attempt is to have $\mathcal{S}$ directly send the ciphertexts to $\mathcal{R}$, and in the sequel, $\mathcal{R}$ tries to decrypt and match. However, this rough idea is problematic since it is insecure even against a semi-honest receiver. Consider $\mathcal{R}$ records the correspondence between x_i and $\mathsf{OKVS}(x_i)$. In this way, $\mathcal{R}$ is able to learn $\mathcal{S}$'s private input y by simple look-up when $y \in X$, rather than merely the fact that $y \in X$. We overcome this difficulty in two steps. The first step is to re-factor the functionality of OPRF to encryption and oblivious decryption functionality. Let $\mathcal{R}$ encrypt the indication string locally. Then $\mathcal{R}$ computes the corresponding OKVS and sends it to $\mathcal{S}$. To ensure the overall protocol still constitutes an RPMT protocol, the second step is to merge the oblivious decryption functionality and PEQT into a new functionality, namely, vector oblivious decryption-then-matching (VODM) functionality. In this functionality, the sender inputs a vector of ciphertexts and the receiver inputs a key and a plaintext. The functionality decrypts these ciphertexts with the key and matches the results with the plaintext input by the receiver. If it matches, the receiver outputs 1, and outputs 0 otherwise.

Putting all the pieces together, we can build mq-RPMT protocol from OKVS, encryption, and VODM functionality in a modular way. (See Section 3 for the technical details).

Two generic constructions of mq-RPMT. Our first generic construction chooses probabilistic SKE as the encryption scheme, and resorts to general 2PC to implement the VODM functionality. See Section 4.1 for details. Our second generic construction chooses re-randomizable PKE as the encryption scheme and uses re-randomization technique to implement VODM functionality, without resorting to generic 2PC.

Our idea is to let $\mathcal{S}$ re-randomize all the ciphertexts and then send the results to $\mathcal{R}$. In this way, $\mathcal{R}$ fulfills the decryption-then-matching functionality in an oblivious manner for all $y_i \in X$. We note that this method will leak some information of $y \notin X$, however, as observed by KRTW, this leakage does not cause any harm to the PSU, since the PSU protocol releases that value anyway.

Looking ahead, one may doubt our PKE-based scheme is inefficient. We note that our PKE-based scheme can still be very efficient because we use PKE techniques in an entirely different way compared to prior PKE-based protocol [KS05, Fri07, DC17]. We only need to perform the encryption, rerandomization, and decryption operations per item, while they need to carry out many ciphertext homomorphism operations per item. See Section 4.2 for details.

Optimization with membership encryption. In the above framework, the underlying encryption schemes must be probabilistic to make the security proof go through. As a consequence, this incurs considerable overhead on communication costs due to ciphertext expansion. Observe that the VODM functionality reveals only one-bit information for every ciphertext. A second thought indicates that a full-fledged encryption scheme might be overkill for our construction of mq-RPMT protocol, and a new type of encryption scheme suffices. We propose the new encryption scheme as membership encryption (ME).

We sketch the definition of membership encryption in the symmetric key setting as below. Let X be a string set. The encryption algorithm takes a key k and an element $x_i \in X$ as inputs, outputs a ciphertext c. The decryption algorithm takes a key k and a ciphertext c as inputs, outputs a bit to indicate if the encrypted element belongs to X. For the correctness, we require that for any $x_i \in X$ and any $c \leftarrow \mathsf{Enc}(k, x_i)$, we have $\mathsf{Dec}(k, c) = 1$. The security requirement is multi-element

pseudorandomness, namely, $\{\mathsf{Enc}(k, x_i)\}_{x_i \in X}$ are computationally indistinguishable to C^n, i.e. the uniform distribution over ciphertext space. The consistency requirement is that a random ciphertext decrypts to "0" with overwhelming probability.

Membership encryption distills the right functionality we need for an encryption scheme in mq-RPMT protocol. It not only encompasses the constructions from randomized SKE and PKE in a unified manner, but also admits new construction from deterministic SKE, which enjoys compact ciphertext. As we elaborate in Section 4.3, this new construction helps to halve the communication complexity on the receiver side.

1.4 Related Work

We survey existing PSU protocols with security against semi-honest adversaries. Hereafter, unless otherwise declared, we calculate the efficiency by assuming a balanced setting, namely the sets of both sender and receiver are of size n.

Kissner and Song [KS05] proposed the first PSU protocol based on polynomial representations and additively homomorphic encryption (AHE). The polynomial representation of a set is to represent a set by a polynomial f, in which each set item is the root of the polynomial. The main observation of them is that when the set of two parties is represented by polynomials f and g, the root of fg is exactly the union items. Two parties compute the AHE of fg. Before decryption, they execute a reduction step to reduce the degree of roots. Then they decrypt the resulting polynomial and compute the roots to obtain the union. The communication and computation complexity of the protocol are both quadratic to the set size n, and the efficiency is very low due to the use of expensive AHE.

Frikken [Fri07] first uses polynomial representation to represent the receiver's set X as a polynomial f, then encrypt f via AHE, and let the receiver send the resulting polynomial encryption $\mathsf{Enc}(f)$ to the sender. Using additive homomorphic property, the sender computes the encryption of $(y\mathsf{Enc}(f(y)), \mathsf{Enc}(f(y)))$ for all $y \in Y$ and sends back to the receiver. The receiver decrypts these ciphertexts. As we can see, $y \in X$ if and only if $f(y) = 0$, receiver obtains two ciphertexts of 0, which contain no information about y. For $y \notin X, f(y) \neq 0$, the receiver decrypts $(y\mathsf{Enc}(f(y)), \mathsf{Enc}(f(y)))$ and computes $y := yf(y) \cdot (f(y))^{-1}$. The communication of this PSU protocol is linear with input size $O(n)$, however, the computation cost is expensive due to the multi-point evaluation on the encrypted polynomial, which is $O(n \log \log n)$.

Davidson and Cid [DC17] proposed a linear communication PSU based on Bloom Filter (BF) and AHE. The receiver first computes the BF of its input set X with λ hash functions and XORs the all-1 string with BF. Then he encrypts this reversed BF with an AHE and sends it to the sender. For each item y in the sender's set, the sender computes λ positions of BF with the same hash functions, and then adds the ciphertexts of corresponding positions. Let c denote the sum of these ciphertexts, the sender computes (c, yc) and sends it back to the receiver. The receiver decrypts these ciphertexts. Note that for $y \in X$, all the λ positions of BF is 0, the receiver obtains $(0, 0)$. For $y \notin X$, the ciphertext c is not an encryption of 0, the receiver could obtain the corresponding y. The communication of this PSU protocol is also linear with input size $O(n)$. However, the computation cost is $O(\lambda)$ public-key operations per item. The total computation is $O(\lambda n)$, which leads to low efficiency.

Garimella et al. [GMR+21] recently proposed a new PSU protocol based on oblivious switching. The main construction of them is a permuted characteristic functionality $\mathcal{F}_{\mathsf{pc}}$. In this functionality, the sender inputs the set $X = \{x_1, \ldots, x_n\}$ and gets a permutation π over $[n]$ as the output. The receiver inputs the set Y and gets a vector $e \in \{0, 1\}^n$, where $e_i = 1$ if $x_{\pi(i)} \in Y$ and $e_i = 0$ otherwise. After that, both parties invoke n instances of OT to let the receiver obliviously retrieve items outside Y. Their core construction of $\mathcal{F}_{\mathsf{pc}}$ needs an oblivious switching network (OSN) subprotocol [MS13]. However, this OSN protocol also leads to a super-linear $O(n \log n)$ communication. In their construction, the receiver has to compute a degree-$3n$ interpolation polynomial. By using the hash to bin technology, the computational complexity is $O(n \log n)$.

Other PSU protocols focus on multi-party settings [KS05, HKK+11, BA12, SCK12], malicious settings [Fri07, HN10, SCK12] and computation with untrusted third party's help [Bf12, CPPT14, SM18]. All of the above constructions rely heavily on expensive AHE or zero-knowledge proof, which are out of the scope of our consideration.

Table 1 provides an asymptotic comparison of our design with the previous PSU works; we elaborate our protocol design in Section 4 and 5.

Table 1: Asymptotic communication and computation costs of two-party PSU protocols in the semi-honest setting.

Protocol	Communication (bits)	Computation
[KS05]	$O(\kappa^3 n^2)$	$O(n^2)$ pub
[Fri07]	$O(\kappa n)$	$O(n \log\log n)$ pub
[DC17]	$O(\kappa\lambda n)$	$O(\lambda n)$ pub
[KRTW19]	$O(\kappa n \log n)$	$O(n \log n \log\log n)$ sym
[GMR+21]	$O(\kappa n \log n)$	$O(n \log n)$ sym
Our SKE-based	$O((\kappa + t)n)$	$O(tn)$ sym
Our PKE-based	$O(\kappa n)$	$O(n)$ pub

Pub: public-key operations; sym: symmetric cryptographic operations. n is the size of the parties' input sets. κ and λ are computational and statistical security parameter respectively (typically $\kappa = 128$ and λ=40). t is the number of AND gate in an SKE decryption circuit. We ignore the pub-key cost of κ base OTs.

2 Preliminaries

2.1 Notation

We denote the parties as receiver $\mathcal{R}$ and sender $\mathcal{S}$, and their respective input sets as X and Y with $|X| = n_{\mathsf{x}}$ and $|Y| = n_{\mathsf{y}}$. In the balanced setting, we often just assume that $n = n_{\mathsf{x}} = n_{\mathsf{y}}$. 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 $\mathbb{F}_{2^\sigma}$ to denote finite field composed of all σ-long bit strings. 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 $f(\kappa) = \mathsf{negl}(\kappa)$. We use the abbreviation PPT to denote probabilistic polynomial-time. By $a \xleftarrow{\text{R}} A$, 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

This work, similar to most protocols for private set operation, operates in the *semi-honest model*, where 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. PSU protocols for the malicious setting exist, e.g., [KS05, Fri07, HN10, BA12, SCK12], but they are less efficient than protocols for the semi-honest setting.

Malicious model is of course the ultimate goal in this line of research. At the same time, we believe semi-honest guarantee is sufficient in many scenarios. The semi-honest adversary model is appropriate for scenarios where execution of the intended software is guaranteed via software attestation or business restrictions, and yet an untrusted third party is able to obtain the transcript of the protocol after its execution, by stealing it or by legally enforcing its disclosure.

Semi-honest security. We use the standard security definition for two-party computation [Can01] in this work.

Definition 1. *Let* $\mathsf{view}^{\Pi}_{\mathcal{S}}(X, Y)$ *and* $\mathsf{view}^{\Pi}_{\mathcal{R}}(X, Y)$ *be the views of* $\mathcal{S}$ *and* $\mathcal{R}$ *in the protocol, and let* $\mathsf{output}(X, Y)$ *be the output of both parties in protocol. A protocol* Π *is said to securely compute functionality* f *in the semi-honest model if for every PPT adversary* $\mathcal{A}$ *there exists a PPT simulator* $\mathsf{Sim}_{\mathcal{S}}$ *and* $\mathsf{Sim}_{\mathcal{R}}$ *such that for all inputs* X *and* Y*,*

$$\{\mathsf{view}^{\Pi}_{\mathcal{S}}(X, Y), \mathsf{output}(X, Y)\} \approx_c \{\mathsf{Sim}_{\mathcal{S}}(X, f(X, Y)), f(X, Y)\}$$

$$\{\mathsf{view}^{\Pi}_{\mathcal{R}}(X, Y), \mathsf{output}(X, Y)\} \approx_c \{\mathsf{Sim}_{\mathcal{R}}(Y, f(X, Y)), f(X, Y)\}$$

2.3 Encryption Schemes

Our construction requires some encryption schemes. We use the standard definition of symmetric-key encryption (SKE) and re-randomizable public-key encryption (ReRand-PKE) schemes. For our purpose, we require a case-tailored security notion called *single-message multi-ciphertext pseudorandomness*. Due to the space limit, we give these definitions in Appendix A.

2.4 Oblivious Transfer

Oblivious transfer (OT) [Rab05] is an important cryptographic primitive used in various multiparty computation protocols.

We define the generalized primitive of 1-out-of-2 OT in Fig. 2.

Parameters: Sender $\mathcal{S}$, Receiver $\mathcal{R}$, message length κ
Functionality:

- Wait for input $b \in \{0, 1\}$ from the receiver $\mathcal{R}$.
- Wait for input (x_0, x_1) from the sender $\mathcal{S}$.
- Give x_b to the receiver $\mathcal{R}$.

Fig. 2: 1-out-of-2 Oblivious Transfer Functionality $\mathcal{F}_{\mathsf{ot}}$

2.5 Oblivious Key-Value Stores

A key-value store [PRTY20, GPR+21] is simply a data structure that maps a set of keys to corresponding values. The definition is as follows:

Definition 2 (Key-Value Store). *A key-value store is parameterized by a set $\mathcal{K}$ of keys, a set $\mathcal{V}$ of values, and a set of function H, and consists of two algorithms:*

- $\mathsf{Encode}_H(\{(x_1, y_1), \ldots, (x_n, y_n)\})$*: on input key-value pairs* $\{(x_i, y_i)\}_{i\in[n]} \subseteq \mathcal{K} \times \mathcal{V}$*, outputs an object D (or, with statistically small probability, an error indicator $\perp$).*
- $\mathsf{Decode}_H(D, x)$ *: on input D and a key x, outputs a value $y \in \mathcal{V}$.*

Correctness. For all $A \subseteq \mathcal{K} \times \mathcal{V}$ with distinct keys:

$$(x, y) \in A \text{ and } \perp \neq D \leftarrow \mathsf{Encode}_H(A) \Longrightarrow \mathsf{Decode}_H(D, x) = y$$

Obliviousness. For all distinct $\{x_1^0, \ldots, x_n^0\}$ and all distinct $\{x_1^1, \ldots, x_n^1\}$, if Encode_H does not output $\perp$ for $\{x_1^0, \ldots, x_n^0\}$ or $\{x_1^1, \ldots, x_n^1\}$, then the distribution of $\{D | y_i \leftarrow \mathcal{V}, i \in [n], \mathsf{Encode}_H((x_1^0, y_1), \ldots, (x_n^0, y_n))\}$ is computationally indistinguishable to $\{D | y_i \leftarrow \mathcal{V}, i \in [n], \mathsf{Encode}_H((x_1^1, y_1), \ldots, (x_n^1, y_n))\}$.

A key-value store is an oblivious key-value store (OKVS) if it satisfies the obliviousness property.

Intuitively, obliviousness means that when value is randomly selected, the distribution of D is independent from key's set. In addition, our application requires OKVS to meet the *Randomness* property defined below to argue the correctness of our scheme.

Randomness. For any $A = \{(x_1, y_1), \ldots, (x_n, y_n)\}$ and $x^* \notin \{x_1, \ldots, x_n\}$, the output of $\mathsf{Decode}_H(D, x^*)$ is statistically indistinguishable to that of uniform distribution over $\mathcal{V}$, where $D \leftarrow \mathsf{Encode}_H(A)$.

The efficiency of an OKVS scheme can be measured by following three parameters:

- **Rate:** Let ratio n/m be the rate of key-value store, where m is the size of object D. Note that the optimal rate is 1.
- **Encoding complexity:** The computational cost of the Encode_H algorithm, as a function of the number n of key-value pairs.
- **Decoding complexity:** The computational cost of the Decode_H algorithm.

We investigated the existing schemes and found that the main candidates for OKVS are: Polynomial, Garbled Bloom Filter (GBF) [DCW13] and Garbled Cuckoo Table (GCT) [PRTY20, RS21, GPR+21] etc. We give the general introduction and detailed comparisons of above OKVS in Appendix B.1.

Before instantiation, 3H-GCT recently proposed by Garimella et al. [GPR+21] could be a good candidate, which has linear encoding complexity $O(n)$ and a rate of 0.81. However, we find that the original 3H-GCT did not meet the *Randomness* we defined before because it was set to 0 in some positions of D. To solve this problem, a natural idea is to set random values in these positions like [RS21] does. We call this modified 3H-GCT as 3H-GCT++ and give the formal description in Fig. 3. We give a proof that our 3H-GCT++ satisfies obliviousness and randomness in Appendix B.2.

2.6 Private Set Union

PSU is a special case of secure two-party computation. The ideal functionality for PSU is given in Fig. 4.

3 Multi-Query Reverse Private Membership Test

3.1 Definition

We propose mq-RPMT and give the formal definition of mq-RPMT functionality in Fig. 5. For generality we set $|Y| = n_{\mathsf{y}}$ and $|X| = n_{\mathsf{x}}$ in our definition.

We define the vector oblivious decryption-then-matching $\mathcal{F}_{\mathsf{vodm}}$ corresponding to encryption scheme $\mathcal{E} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ in Fig. 6, as a component of mq-RPMT.

3.2 Framework of Multi-Query RPMT

Now we describe our framework of mq-RPMT protocol. As we mentioned in Section 1.3, the cryptographic primitives we use are a single-message multi-ciphertext pseudorandomness encryption scheme $\mathcal{E} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$, an OKVS scheme $(\mathsf{Encode}_H, \mathsf{Decode}_H)$ and $\mathcal{F}_{\mathsf{vodm}}$ functionality.

Let $Y = \{y_1, \ldots, y_{n_{\mathsf{y}}}\}$ and $X = \{x_1, \ldots, x_{n_{\mathsf{x}}}\}$ be the input of mq-RPMT sender $\mathcal{S}$ and receiver $\mathcal{R}$. First, the receiver $\mathcal{R}$ picks an indication string s[5]. Then $\mathcal{R}$ chooses a random key k used in encryption scheme $\mathcal{E}$ to encrypt s for n_{x} times, and obtains $(s_1, \ldots, s_{n_{\mathsf{x}}})$. Next, $\mathcal{R}$ computes an OKVS $D := \mathsf{Encode}_H((x_1, s_1), \ldots, (x_{n_{\mathsf{x}}}, s_{n_{\mathsf{x}}}))$ and sends D to $\mathcal{S}$. After receiving D, $\mathcal{S}$ computes $s_i^* = \mathsf{Decode}_H(D, y_i)$ for $i \in [n_{\mathsf{y}}]$. Now $\mathcal{S}$ and $\mathcal{R}$ invoke the VODM functionality $\mathcal{F}_{\mathsf{vodm}}$. $\mathcal{S}$ acts as sender with input $S = \{s_1^*, \ldots, s_{n_{\mathsf{y}}}^*\}$ and $\mathcal{R}$ acts as receiver with input (k, s). As a result, $\mathcal{S}$ receives nothing and $\mathcal{R}$ receives $b \in \{0,1\}^{n_{\mathsf{y}}}$, satisfying $b_i = 1$ if and only if s_i^* decrypts to s. Now, we give our framework of mq-RPMT protocol in Fig. 7.

Correctness. For all $i \in [n_{\mathsf{y}}]$, if $y_i \in X$, there is an $x_j \in X, j \in [n_{\mathsf{x}}]$ s.t. $y_i = x_j$. In this case, $s_i^* = \mathsf{Decode}_H(D, h(x_j)) = s_j$. Since $s_j = \mathsf{Enc}(k, s)$, we have $\mathsf{Dec}(k, s_j) = s$, which means $b_i = 1$. In the case $y_i \notin X$, if hash functions collide, that is, $h(y_i) = h(x)$ for some $y_i \notin X$, the correctness will be violated. By setting $\sigma = \lambda + \log n_{\mathsf{x}} n_{\mathsf{y}}$, a union bound shows probability of collision is negligible $2^{-\lambda}$. When no collision occurs, from the randomness of OKVS, $s_i^* = \mathsf{Decode}_H(D, h(y_i))$ is a random ciphertext, resulting in s_i^* is not the encryption of s with overwhelming probability. The union bound guarantees that for all $y_i \notin X$, the probability that there exists an s_i^* s.t. $\mathsf{Dec}(k, s_i^*) = s$ is negligible.

We now state and prove the security properties of our mq-RPMT.

Theorem 1. *Assume the encryption scheme $\mathcal{E} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ satisfies single-message multi-ciphertext pseudorandomness. The protocol in Fig. 7 securely computes $\mathcal{F}_{\mathsf{mq\text{-}rpmt}}$ against semi-honest adversaries in the $\mathcal{F}_{\mathsf{vodm}}$-hybrid model.*

Proof. Due to space limitation, we only sketch here the simulators for the two cases of corrupt $\mathcal{S}$ and corrupt $\mathcal{R}$, the full security proof (via hybrid arguments) is deferred to Appendix C.

[5] In fact, our indication string s could be any fixed value, e.g. $s = 0$, while s in KRTW must be selected randomly.

Parameters:

- Computational security parameter κ and statistical security parameter λ.
- Input length n.
- A finite group $\mathbb{G}$.
- Random fuctions $h_1, h_2, h_3 : \{0,1\}^* \to [m']$ and $r : \{0,1\}^* \to \{0,1\}^{d+\lambda}$.
- Parameters $m' = 1.3n$ and $d = 0.5 \log n$, as shown in [GPR+21], where d upper bound the size of 2-core of a (m', n)-Cuckoo graph.
- Output length $m = m' + d + \lambda$.

$\mathsf{Encode}_H(\{(x_1, y_1), \ldots, (x_n, y_n)\})$:

1. Define $l(x) \in \{0,1\}^{m'}$ to be all zeroes except 1s at positions $h_1(x)$, $h_2(x), h_3(x)$. Here we assume the weight of $l(x)$ is 3. Let $\mathrm{row}(x) := l(x)||r(x)$,

$$M^{(0)} = \begin{bmatrix} l(x_1) \\ \vdots \\ l(x_n) \end{bmatrix} \in \{0,1\}^{n \times m'}, M^{(1)} = \begin{bmatrix} r(x_1) \\ \vdots \\ r(x_n) \end{bmatrix} \in \{0,1\}^{n \times (d+\lambda)}$$

   and let

$$M = M^{(0)}||M^{(1)} = \begin{bmatrix} \mathrm{row}(x_1) \\ \vdots \\ \mathrm{row}(x_n) \end{bmatrix} \in \{0,1\}^{n \times m}.$$

2. Initialize empty vectors $D_L \in \mathbb{G}^{m'}$ and $D_R \in \mathbb{G}^{d+\lambda}$, let $D = D_L||D_R$.
3. Initialize stack P.
4. While there is a node $j \in [m']$ such that the set $\{x_i \notin P | j \in \{h_1(x_i), h_2(x_i), h_3(x_i)\}\}$ is a singleton: Let x_i be the element of that singleton, and push x_i onto P.
5. Let $S = \{x_i | x_i \notin P\}$, and let $R \subset [n]$ index the rows of M in S, i.e. $R = \{i | M^{(0)}_{i,h_1(x_i)} = M^{(0)}_{i,h_2(x_i)} = M^{(0)}_{i,h_3(x_i)} = 1 \wedge x_i \in S\}$. Let $\tilde{d} := |R|$ and abort if $\tilde{d} > d$.
6. Let $\tilde{M}^{(1)} \in \{0,1\}^{\tilde{d} \times (d+\lambda)}$ be the submatrix of $M^{(1)}$ obtained by taking the row indexed by R. Abort if $\tilde{M}^{(1)}$ does not contain an invertible $\tilde{d} \times \tilde{d}$ matrix. Otherwise let $\tilde{M}^*$ be one such matrix and $C \subset [d+\lambda]$ index the corresponding columns of $\tilde{M}^{(1)}$.
7. Let $C' := \{j | i \in R, M^{(0)}_{i,j} = 1\} \cup ([d+\lambda] \setminus C + m')$ and for $i \in C'$ assign $D_i \leftarrow \mathbb{G}$. For $i \in R$, define $y'_i := y_i - (MD^T)_i$ where D_i is assumed to be zero if unssigned.
8. Using Gaussian elimination solve the system $\tilde{M}^*(D_{m'+C_1}, \ldots, D_{m'+C_{\tilde{d}}})^T = (y'_{R_1}, \ldots, y'_{R_{\tilde{d}}})^T$.
9. While P not empty:
   (a) pop x_i from P.
   (b) D_L is undefined in at least one of the positions $h_1(x_i), h_2(x_i), h_3(x_i)$. Set the undefined position(s) so that $\langle \mathrm{row}(x_i), D \rangle = y_i$.
10. Set any empty position in D with a random value from $\mathbb{G}$.
11. Output D.

$\mathsf{Decode}_H(D, x)$:

1. Return $\langle \mathrm{row}(x), D \rangle$.

Fig. 3: 3H-GCT++ algorithm

Parameters: Sender $\mathcal{S}$, Receiver $\mathcal{R}$, set sizes n_{y} and n_{x}.
Functionality:

- Wait for input $X = \{x_1, \ldots, x_{n_{\mathsf{x}}}\} \subset \{0,1\}^*$ from the receiver $\mathcal{R}$.
- Wait for input $Y = \{y_1, \ldots, y_{n_{\mathsf{y}}}\} \subset \{0,1\}^*$ from the sender $\mathcal{S}$.
- Give output $X \cup Y$ to the receiver $\mathcal{R}$.

Fig. 4: Private Set Union Functionality $\mathcal{F}_{\mathsf{psu}}$

Parameters: Sender $\mathcal{S}$, Receiver $\mathcal{R}$, set sizes n_{y} and n_{x}
Functionality:

- Wait for input $Y = \{y_1, \ldots, y_{n_{\mathsf{y}}}\} \subset \{0,1\}^*$ from the sender $\mathcal{S}$.
- Wait for input $X = \{x_1, \ldots, x_{n_{\mathsf{x}}}\} \subset \{0,1\}^*$ from the receiver $\mathcal{R}$.
- Set $b_i = 1$ if and only if $y_i \in X$ and $b_i = 0$ otherwise for $i \in [n_{\mathsf{y}}]$. Give output $b \in \{0,1\}^{n_{\mathsf{y}}}$ to the receiver $\mathcal{R}$.

Fig. 5: Multi-Query Reverse Private Membership Test Functionality $\mathcal{F}_{\mathsf{mq\text{-}rpmt}}$

Corrupt sender: To simulate OKVS in Step 3, the simulator computes a random OKVS D by selecting n_{x} random key-value pairs. Then, the simulator sets $s_i^* := \mathsf{Decode}_H(D, h(y_i))$ and invokes underlying VODM simulator with inputs $(s_1^*, \ldots, s_{n_{\mathsf{y}}}^*)$.

Briefly, this simulation is indistinguishable for the following reasons: the single-message multi-ciphertext pseudorandomness of the encryption ensures that value (ciphertext) is indistinguishable from random, and then by the obliviousness of OKVS, D is distributed uniformly.
Corrupt receiver: The simulator for a corrupt $\mathcal{R}$ first obtains b from the ideal mq-RPMT functionality. The only message that needs to be simulated is the VODM functionality in Step 5. The simulator just executes Step 1 honestly and invokes the underlying VODM simulator with inputs (k, s, b).

4 Generic Constructions of Multi-Query RPMT

In this section, we give two generic constructions of mq-RPMT protocol. In the first construction, we use SKE as the encryption scheme and generic 2PC to implement VODM. The advantage is that this scheme only uses OT and symmetric operations. In the second construction, we use PKE and a re-randomization method to implement the encryption scheme and a leaky version of VODM respectively, which leads to a leaky version of mq-RPMT. However, as observed by KRTW, this leaky version can still be used to construct a secure PSU. Both schemes achieve linear computation and communication complexity.

4.1 Construction from SKE and 2PC

As we noted before, a single-message multi-ciphertext pseudorandom SKE and 2PC are sufficient for constructing mq-RPMT. The correctness and security can be directly derived from the general construction in Section 3.2. It is straightforward to show that PRF-based SKE satisfies the single-message multi-ciphertext pseudorandomness property. We give proof in the Appendix D for completeness.

We use the general 2PC as the implementation of VODM. Formally,

Theorem 2. *Taking the PRF-based SKE as the encryption scheme in Fig. 7. Assuming that the 2PC implementing VODM is semi-honest secure, then the protocol in Fig. 7 securely computes $\mathcal{F}_{\mathsf{mq\text{-}rpmt}}$ against semi-honest adversaries.*

This theorem immediately follows from Lemma 1 and Theorem 1.

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

Parameters:

- Two parties: sender $\mathcal{S}$ and receiver $\mathcal{R}$.
- A single-message multi-ciphertext pseudorandomness encryption scheme $\mathcal{E} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$.
- Ideal $\mathcal{F}_{\mathsf{vodm}}$ primitives specified in Fig. 6.
- An OKVS scheme $(\mathsf{Encode}_H, \mathsf{Decode}_H)$.
- A collision-resistant hash function $h(x) : \{0,1\}^* \to \{0,1\}^\sigma$.

Input of $\mathcal{S}$: $Y = \{y_1, \ldots, y_{n_\mathsf{y}}\} \subset \{0,1\}^*$
Input of $\mathcal{R}$: $X = \{x_1, \ldots, x_{n_\mathsf{x}}\} \subset \{0,1\}^*$
Protocol:

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

Fig. 7: General construction of mq-RPMT Protocol $\Pi_{\mathsf{mq\text{-}rpmt}}$

4.2 Construction from Re-randomizable PKE

Now we consider a specialized way to construct $\mathcal{F}_{\mathsf{vodm}}$. Our main idea is that since the receiver cannot know the randomness used in each ciphertext, as long as the encryption scheme satisfies the property of rerandomization, the sender can re-randomize all ciphertexts and send the new ciphertexts to the receiver so that the receiver can not obtain additional information by comparing randomness. Note that another problem arises here. The property of re-randomization can only guarantee that for $y \in X$, the receiver is not allowed to learn which one is the sender's element. For $y \notin X$, the ciphertext s_i^* obtained by the sender is related to y, so the plaintext obtained by the receiver is also related to y, which will reveal the information of y. However, as observed by KRTW, in the case of $y \notin X$, we want (in the overall PSU protocol) the receiver to learn y anyway. Fully secure mq-RPMT is actually overkill for PSU, a relaxed version suffices. We define the leaky VODM functionality in Fig. 8.

Since the SKE scheme is hard to re-randomize, we consider the use of public-key encryption (PKE) which is easier to re-randomize. We describe our PKE-based leaky VODM protocol in Fig. 9.

We now state and prove the security of the above leaky VODM protocol.

Theorem 3. *Assume the security of the ReRand-PKE scheme. The protocol in Fig. 9 securely computes $\mathcal{F}_{\mathsf{lvodm}}$ against semi-honest adversaries.*

Proof. Because the sender does not receive messages in the protocol, we just need to simulate the view of the receiver. We exhibit simulator $\mathsf{Sim}_\mathcal{R}$ for simulating corrupt $\mathcal{R}$.

Corrupt receiver: $\mathsf{Sim}_\mathcal{R}(pk, sk, s, b, \{s_i' | b_i = 0\})$ simulates the view of corrupt semi-honest receiver. Note that the only messages that need to be simulated by the simulator are ciphertexts $\{\bar{s_i}\}_{i \in [n]}$.

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 $b \in \{0,1\}^n$ and $\{s_i' | b_i = 0\}$ to the receiver $\mathcal{R}$.

Fig. 8: Leaky VODM Functionality $\mathcal{F}_{\mathsf{lvodm}}$

Parameters:

- Two parties: sender $\mathcal{S}$ and receiver $\mathcal{R}$.
- A re-randomizable PKE scheme $(\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{ReRand})$.

Input of $\mathcal{S}$: $(pk, S^* = \{s_1^*, \ldots, s_n^*\})$
Input of $\mathcal{R}$: $((pk, sk), s)$
Protocol:

1. $\mathcal{S}$ selects random $r_1', \ldots, r_n'$ and computes $\bar{s_i} := \mathsf{ReRand}(pk, s_i^*; r_i')$ for $i \in [n]$.
2. $\mathcal{S}$ sends $\bar{s_1}, \ldots, \bar{s_n}$ to $\mathcal{R}$.
3. $\mathcal{R}$ sets $b_i = 1$ if and only if $\mathsf{Dec}(sk, \bar{s_i}) = s$ for $i \in [n]$.

Fig. 9: PKE-based Leaky VODM Protocol Π_{lvodm}

$\mathsf{Sim}_{\mathcal{R}}$ computes $\bar{s_i} := \mathsf{Enc}(pk, s; r_i)$ if $b_i = 1$ and $\bar{s_i} := \mathsf{Enc}(pk, s_i'; r_i)$ if $b_i = 0$ for $i \in [n]$. $\mathsf{Sim}_{\mathcal{R}}$ appends $\{\bar{s_i}\}_{i \in [n]}$ to the view.

The indistinguishability of ReRand-PKE scheme guarantees the view output by $\mathsf{Sim}_{\mathcal{R}}$ is indistinguishable from the real one.

Note that the mq-RPMT constructed with the above leaky VODM is also a leaky version. We don't give a specific description of this leaky mq-RPMT. Instead, we use leaky VODM to construct PSU protocol directly and prove its security in Appendix E.

4.3 Unification with Membership Encryption

We have presented two generic constructions of mq-RPMT protocols from probabilistic SKE and probabilistic PKE respectively. It is intriguing to study if there is a unified way to encompass the two different constructions.

We retrospect the high level idea underlying our mq-RPMT protocol. If privacy is not a concern, reverse membership test can be simply done by having the receiver first create a membership relation R for his set Y, namely $\mathsf{R}(y) = 1$ iff $y \in Y$, then having the sender send his elements to the receiver in clear. To make the reverse membership test private, the receiver can "encrypt" his membership relation and send the "encoding" of resulting ciphertexts to the sender. After receiving the "encoding", the sender is able to retrieve the membership encryptions corresponding to his elements. In the sequel, the receiver can fulfill the reverse private membership test by decrypting the ciphertexts in an oblivious manner.

Based on the above discussion, we realize that the right encryption scheme needed in our mq-RPMT protocol is an abstract new notion called *membership encryption (ME)*. Roughly speaking, ME for set X encrypts an element x into a ciphertext, which decrypts to "1" if $x \in X$. We formalize the syntax and security notion of ME in the private-key setting as below.

Definition 3 (Membership Encryption). *Membership encryption for set X consists of four polynomial time algorithms satisfying the following properties.*

- $\mathsf{Setup}(1^\kappa)$*: on input a security parameter κ, outputs public parameters pp, which include the ciphertext space C.*

- $\mathsf{KeyGen}(pp, X)$*: on input public parameters pp and* $X \subseteq \{0,1\}^*$*, outputs a key* k*.*
- $\mathsf{Enc}(k, x)$*: on input a key* k *and an element* $x \in X$*, outputs a ciphertext* $c \in C$*. For uttermost generality, the behavior of* Enc *on* $x \notin X$ *is unspecified. Looking ahead, such treatment suffices for the construction of mq-RPMT protocol.*
- $\mathsf{Dec}(k, c)$*: on input a key* k *and a ciphertext* $c \in C$*, outputs "1" indicating* c *is an encryption of an element* x *in* X *and "0" if not.*

Correctness. For any $x \in X$, $\forall k \leftarrow \mathsf{KeyGen}(pp, X)$, $\mathsf{Dec}(k, c = \mathsf{Enc}(k, x)) = 1$.
Consistency. For any $x \notin X$, $\Pr[\mathsf{Dec}(k, c) = 0] = 1 - \epsilon(\kappa)$, where $pp \leftarrow \mathsf{Setup}(1^\kappa)$, $k \leftarrow \mathsf{KeyGen}(pp, X)$, $c \xleftarrow{\mathrm{R}} C$. Here, ϵ represents the consistency error, which must be negligible in κ.
Multi-element pseudorandomness. For any n distinct elements $x_1, \ldots, x_n \in X$, $\{\mathsf{Enc}(k, x_i)\}_{i \in [n]} \approx_c U_{C^n}$.

The ME notion naturally extends to the public-key setting by letting the KeyGen algorithm generate a keypair (pk, sk), in which pk is used to encrypt and sk is used to decrypt. We omit the details for its straightforwardness.

We then study the generic construction of ME. Note that the essence of ME is to encrypt element's membership relation, rather than the element itself. The membership relation can be created by establishing a mapping H from elements to the set under test. Basically, there are two extreme cases of mapping. The first is to select a single indication string s as the characteristic of the set, then map all elements to s, i.e., $\mathsf{H} : x_i \to s$, which we refer to as *lossy mapping*. The second is to select n indication strings s_i as the characteristic of the set, then map elements to distinct indication strings, i.e., $\mathsf{H} : x_i \to s_i$, which we refer to as *injective mapping*. With the above understanding in head, we present various constructions of ME by mixing encryption schemes and membership mapping.

ME from probabilistic SKE and lossy mapping. The construction is as below.

- $\mathsf{Setup}(1^\kappa)$: runs SKE.$\mathsf{Setup}(1^\kappa)$ to generate pp.
- $\mathsf{KeyGen}(pp, X)$: runs SKE.$\mathsf{KeyGen}(pp)$ to sample k_{ske}, picks a random element $s \in M$, where M is the message space of SKE, sets H be a mapping that maps all elements in X to s, outputs $k = (k_{\mathrm{ske}}, \mathsf{H})$
- $\mathsf{Enc}(k, x)$: parses $k = (k_{\mathrm{ske}}, \mathsf{H})$, outputs $c \leftarrow$ SKE.$\mathsf{Enc}(k_{\mathrm{ske}}, \mathsf{H}(x))$.
- $\mathsf{Dec}(k, c)$: parses $k = (k_{\mathrm{ske}}, \mathsf{H})$, outputs '1' iff SKE.$\mathsf{Dec}(k_{\mathrm{ske}}, c) = s$.

ME from probabilistic PKE and lossy mapping. The construction is as below.

- $\mathsf{Setup}(1^\kappa)$: runs PKE.$\mathsf{Setup}(1^\kappa)$ to generate pp.
- $\mathsf{KeyGen}(pp, X)$: runs PKE.$\mathsf{KeyGen}(pp)$ to generate $(pk_{\mathrm{pke}}, sk_{\mathrm{ske}})$, picks a random element $s \in M$, where M is the message space of PKE, sets H be a mapping that maps all elements in X to s, outputs $pk = pk_{\mathrm{pke}}$ and $sk = (sk_{\mathrm{pke}}, \mathsf{H})$
- $\mathsf{Enc}(pk, x)$: parses $pk = pk_{\mathrm{pke}}$, outputs $c \leftarrow$ PKE.$\mathsf{Enc}(pk_{\mathrm{pke}}, \mathsf{H}(x))$.
- $\mathsf{Dec}(sk, c)$: parses $sk = (sk_{\mathrm{pke}}, \mathsf{H})$, outputs '1' iff PKE.$\mathsf{Dec}(sk_{\mathrm{pke}}, c) = s$.

Theorem 4. *If SKE (resp. PKE) satisfies single-message multi-ciphertext pseudorandomness, then the above ME construction satisfies multi-element pseudorandomness with consistency error* $1/|M|$*.*

The above ME constructions are exactly the backbones of our generic constructions of mq-RPMT protocol presented in Section 4.1 and 4.2. Since ME requires multi-element pseudorandomness, the use of lossy mapping inherently stipulates that the accompanying encryption schemes are probabilistic. Therefore, in this case the ciphertext expansion is unavoidable. For example, in PRF-based probabilistic SKE, the length of ciphertext is twice that of plaintext. In the design of our mq-RPMT protocol, the value in OKVS is exactly ciphertext. As a consequence, ciphertext expansion incurs overhead to the size of OKVS and thus also the communication cost on the receiver side. For this reason, reducing the ciphertext expansion factor will immediately improve the performance of the overall mq-RPMT protocol.

An important observation is that if we switch to injective mapping, then ME can be built from deterministic encryption schemes satisfying *multi-message multi-ciphertext pseudorandomness*. The constructions are similar as above except the decryption algorithm outputs '1' iff the decryption

result falls into the prior-fixed indication string set $S = \{s_i\}_{i\in[n]}$. In instantiation, we take $\mathsf{H} : x_i \to i$ as the membership mapping, which renders efficient membership decryption by testing whether the decryption is less than n.

Formally, we have the following theorem:

Theorem 5. *If SKE (resp. PKE) satisfies multi-message multi-ciphertext pseudorandomness, then the ME construction satisfies multi-element pseudorandomness with consistency error $n/|M|$.*

If we instantiate the ME from the PRP-based deterministic SKE and injective mapping, the ciphertext expansion factor is optimal. Therefore, a drop-in replacement to the ME from PRF-based probabilistic SKE and lossy mapping will reduce the size of OKVS in the mq-RPMT protocol by half.

Due to space constraints, we put the description that how to construct mq-RPMT using the language of ME in the Appendix F.

5 Our PSU Protocol

In this section, we describe our PSU construction achieving linear complexity and prove its semi-honest security.

5.1 Generic Construction of PSU Protocols

With mq-RPMT and OT, we can simply combine them to construct a PSU protocol. We give the formal description in Fig. 10.

Parameters:

- Two parties: sender $\mathcal{S}$ and receiver $\mathcal{R}$.
- Ideal $\mathcal{F}_{\mathsf{mq\text{-}rpmt}}$ and $\mathcal{F}_{\mathsf{ot}}$ primitives specified in Fig. 5, and Fig. 2, respectively.

Input of $\mathcal{S}$: $Y = \{y_1, \ldots, y_{n_\mathsf{y}}\} \subset \{0,1\}^*$
Input of $\mathcal{R}$: $X = \{x_1, \ldots, x_{n_\mathsf{x}}\} \subset \{0,1\}^*$
Protocol:

1. $\mathcal{S}$ and $\mathcal{R}$ invoke the mq-RPMT functionality $\mathcal{F}_{\mathsf{mq\text{-}rpmt}}$ first. The sender $\mathcal{S}$ acts as the sender in mq-RPMT with input Y and receives nothing. The receiver $\mathcal{R}$ acts as the receiver in mq-RPMT with input X and receives $b \in \{0,1\}^{n_\mathsf{y}}$.
2. $\mathcal{R}$ initialize set $Z := \{\}$.
3. For $i \in [n_\mathsf{y}]$:
   (a) $\mathcal{S}$ and $\mathcal{R}$ invoke the OT functionality $\mathcal{F}_{\mathsf{ot}}$.
   (b) $\mathcal{S}$ acts as sender with input $(y_i, \perp)$.
   (c) $\mathcal{R}$ acts as receiver with input b_i.
   (d) $\mathcal{R}$ obtains the OT output z_i and sets $Z = Z \cup \{z_i\}$.
4. $\mathcal{R}$ outputs $X \cup Z$.

Fig. 10: Private Set Union Protocol Π_{psu}

We now state and prove the security properties of the above PSU protocol.

Theorem 6. *The protocol in Fig. 10 securely computes $\mathcal{F}_{\mathsf{psu}}$ against semi-honest adversaries in the $(\mathcal{F}_{\mathsf{mq\text{-}rpmt}}, \mathcal{F}_{\mathsf{ot}})$-hybrid model.*

Proof. We exhibit simulators $\mathsf{Sim}_\mathcal{R}$ and $\mathsf{Sim}_\mathcal{S}$ for simulating corrupt $\mathcal{R}$ and $\mathcal{S}$ respectively, and argue the indistinguishability of the produced transcript from the real execution.

Corrupt Sender: $\mathsf{Sim}_\mathcal{S}(Y = \{y_1, \ldots, y_{n_\mathsf{y}}\})$ simulates the view of corrupt semi-honest sender. It executes as follows:

1. $\mathsf{Sim}_\mathcal{S}$ invokes mq-RPMT simulator $\mathsf{Sim}^S_{\mathsf{mq\text{-}rpmt}}(Y)$ and appends the output to the view.

2. For $i \in [n_{\mathsf{y}}]$, $\mathsf{Sim}_{\mathcal{S}}$ invokes OT simulator $\mathsf{Sim}^{S}_{\mathsf{ot}}(y_i, \perp)$ 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.
Corrupt Receiver: $\mathsf{Sim}_{\mathcal{R}}(X = \{x_1, \ldots, x_{n_{\mathsf{x}}}\}, X \cup Y)$ simulates the view of corrupt semi-honest receiver. It executes as follows:

1. $\mathsf{Sim}_{\mathcal{R}}$ defines the set $Z := X \cup Y \setminus X$, i.e. the set of elements that Y "brings to the union". Next, it uses $\perp$ to pads Z to n_{y} elements and permutates these elements randomly. Let $Z = \{z_1, \ldots, z_{n_{\mathsf{y}}}\}$.
2. $\mathsf{Sim}_{\mathcal{R}}$ sets $b_i = 1$ if and only if $z_i \in X$ for $i \in [n_{\mathsf{y}}]$. Then, it invokes mq-RPMT simulator $\mathsf{Sim}^{R}_{\mathsf{mq\text{-}rpmt}}(X, b)$ and appends the output to the view.
3. For $i \in [n_{\mathsf{y}}]$, $\mathsf{Sim}_{\mathcal{R}}$ invokes OT simulator $\mathsf{Sim}^{R}_{\mathsf{ot}}(b_i, z_i)$ and appends the output to the view.

Now we argue that the view output by $\mathsf{Sim}_{\mathcal{R}}$ is indistinguishable from the real one. In the simulation, the way $\mathcal{R}$ obtains the elements in $Z = X \setminus Y$ is identical to the real execution. By the underlying simulators' indistinguishability, the simulated view is computationally indistinguishable from the real one.

5.2 Instantiation of PSU

For our SKE-based construction, we can use a PRP as we mentioned in Section 4.3 to instantiate SKE, which can achieve an optimal ciphertext expansion factor. Since we need to perform the 2PC decryption computation, we use the LowMC [ARS+15] as our PRP instantiation to minimize the circuit size. As for generic 2PC, there are two classical methods, e.g. garbled circuit [Yao86] or GMW [GMW87]. The former has a constant number of rounds, while the latter has a lower communication. Since the communication has a greater impact on our scheme, we consider instantiating 2PC by GMW.

For our PKE-based construction, we use the well-known ECC ElGamal [Gam85] scheme as our ReRand-PKE.

5.3 Communication Cost

Now we analyze the communication cost of our two PSU constructions. For the SKE-based construction, we use our ME optimization in Section 4.3.

Let's first analyze the size of decryption circuit in our SKE-based construction: the circuit needs to compute decryption of every $\{s_i^*\}_{i \in [n_{\mathsf{y}}]}$ and compares the result with n_{x}. If $\mathsf{Dec}(k, s_i^*) < n_{\mathsf{x}}$, it sets $b_i = 1$ and $b_i = 0$ otherwise. The total number of decryption computations is n_{y}. To compare whether a σ long string is less than n_{x}, we only need to compute whether the OR of its first $\sigma - \log n_{\mathsf{x}}$ bits are 1, which requires $\sigma - \log n_{\mathsf{x}} - 1$ AND gates (since $a \vee b = \bar{a} \wedge \bar{b}$). The total number of AND gates is $n_{\mathsf{y}}(t + \sigma - \log n_{\mathsf{x}}) = O(tn_{\mathsf{y}})$, where t is the number of AND gates in a PRP decryption circuit.

Now we are ready to calculate the communication of PSU protocol. Note that the communication of our protocol consists of OKVS, VODM protocol and OT protocol. We analyze their complexity respectively. We use the symbol Φ to represent the communication complexity, and its subscripts represent different components.

- OKVS in both constructions: as we showed in Section 2.5, we use 3H-GCT++ as our OKVS scheme:

$$\Phi_{\mathsf{okvs}}(n_{\mathsf{x}}) = (1.3n_{\mathsf{x}} + d + \lambda)|c|$$

  where $|c|$ is the size of ciphertext, $|c| = \lambda + \log n_{\mathsf{x}} n_{\mathsf{y}}$ and 4κ for SKE-based and PKE-based scheme respectively.
- Oblivious decryption:
  - In SKE-based construction: we use $\Phi^{\mathsf{ske}}_{\mathsf{vod}}(n_{\mathsf{y}}, n_{\mathsf{x}})$ to denote the communication of computing oblivious decryption circuit. As we said in Section 5.2, we use GMW as our 2PC instantiation, the communication consists of *input sharing*, *multiplication gate computation* and *output reconstruction*. In the input sharing phase, the communication is $\kappa + n_{\mathsf{y}}\sigma$ bits, and in the output reconstruction phase, it is n_{y} bits. Using Beaver triple [Bea91], $4n_{\mathsf{y}}(t + \sigma - \log n_{\mathsf{x}})$ bits are needed in multiplication phase. So we have $\Phi^{\mathsf{ske}}_{\mathsf{vod}}(n_{\mathsf{y}}, n_{\mathsf{x}}) = \kappa + n_{\mathsf{y}}\sigma + 4n_{\mathsf{y}}(t + \sigma - \log n_{\mathsf{x}}) + n_{\mathsf{y}}$

- In PKE based construction: the communication of leaky VODM functionality, denoted by $\Phi^{\mathsf{pke}}_{\mathsf{lvodm}}(n_{\mathsf{y}}, n_{\mathsf{x}}) = 4n_{\mathsf{y}}\kappa$

– OT in both constructions:

$$\Phi_{\mathsf{ot}}(n_{\mathsf{y}}) = n_{\mathsf{y}}(\kappa + \sigma).$$

Let $\Phi^{\mathsf{ske}}_{\mathsf{psu}}(n_{\mathsf{y}}, n_{\mathsf{x}})$ denote communication of SKE-based construction and let $\Phi^{\mathsf{pke}}_{\mathsf{psu}}(n_{\mathsf{y}}, n_{\mathsf{x}})$ denote communication of PKE-based construction. The overall communication cost of our PSU protocol is:

$$\Phi^{\mathsf{ske}}_{\mathsf{psu}}(n_{\mathsf{y}}, n_{\mathsf{x}}) = (\lambda + \log n_{\mathsf{x}} n_{\mathsf{y}})(1.3n_{\mathsf{x}} + d + \lambda) + \Phi^{\mathsf{ske}}_{\mathsf{vod}}(n_{\mathsf{y}}, n_{\mathsf{x}}) + n_{\mathsf{y}}(\kappa + \sigma)$$

$$\Phi^{\mathsf{pke}}_{\mathsf{psu}}(n_{\mathsf{y}}, n_{\mathsf{x}}) = 4\kappa(1.3n_{\mathsf{x}} + d + \lambda) + 4\kappa n_{\mathsf{y}} + n_{\mathsf{y}}(\kappa + \sigma).$$

6 Implementation and Performance

Recall that we have presented two variants of our protocol. In this section, we will refer to them as:

– SKE-PSU: PSU protocol with SKE-based mq-RPMT, where SKE and VODM are instantiated with PRP and GMW [GMW87] respectively.
– PKE-PSU: PSU protocol with PKE-based mq-RPMT, where ReRand-PKE is instantiated with ECC ElGamal encryption scheme.

The OKVS instantiation of both schemes uses the 3H-GCT++ in Fig.3. We focus on the case where $n_{\mathsf{y}} = n_{\mathsf{x}} = n$, i.e., both parties have equal-size sets.

6.1 Theoretical Analysis of Communication

In Table 2, we show the theoretical communication complexity of our protocol compared with the Frikken protocol [Fri07], the DC protocol [DC17], the KRTW protocol [KRTW19] and the GMRSS protocol [GMR+21] in the semi-honest setting. This measures how much communication the protocols require on an idealized network where we do not care about the protocol metadata, realistic encodings, byte alignment, etc. In practice, data is split up into multiples of bytes (or CPU words), and different data is encoded with headers, etc. Empirical measurements of such real-world costs are given later in Table 4.

Protocol	Communication	$n = n_{\mathsf{y}} = n_{\mathsf{x}}$		
		2^{14}	2^{18}	2^{22}
Frikken [Fri07]	$N + 2n_{\mathsf{x}}N + 4n_{\mathsf{y}}N$	$12288n$	$12288n$	$12288n$
DC [DC17]	$2\lambda n_{\mathsf{x}}N + 4n_{\mathsf{y}}N$	$172032n$	$172032n$	$172032n$
KRTW [KRTW19]	$\beta u(2\rho + \lambda + (u+2)\sigma) + \beta u(\kappa + \sigma)$	$14977n$	$16927n$	$18956n$
GMRSS [GMR+21]	$1.27n_{\mathsf{y}}\rho + 3n_{\mathsf{x}}\sigma + (1.27n_{\mathsf{y}} \log n_{\mathsf{y}} + n_{\mathsf{y}})(\kappa + \sigma)$	$5417n$	$6687n$	$7947n$
SKE-PSU	$(1.3n_{\mathsf{x}} + d + \lambda)\sigma + \kappa + n_{\mathsf{y}}\sigma$ $+4n_{\mathsf{y}}(t + \sigma - \log n_{\mathsf{x}}) + n_{\mathsf{y}}(\kappa + \sigma)$	$568n$ $+4tn$	$610n$ $+4tn$	$653n$ $+4tn$
PKE-PSU	$4\kappa(1.3n_{\mathsf{x}} + d + \lambda) + 4\kappa n_{\mathsf{y}} + n_{\mathsf{y}}(\kappa + \sigma)$	$1373n$	$1381n$	$1389n$

Table 2: Theoretical communication costs of PSU protocols (in bits), calculated using computational security κ= 128 and statistical security λ= 40. Ignore costs of base OTs (in our protocol, KRTW and GMRSS) which are independent of input size. N is the size of the public key in Pallier encryption scheme (2048 is used here). β and u are the number of bins and maximum bin size respectively. ρ is the width of OT extension matrix (depends on n and protocol). t is the number of AND gates in the PRP decryption circuit. σ is the length of items. n_{y} and n_{x} are the input sizes of the sender and receiver respectively.

For set sizes in the range 2^{14} to 2^{22}, our PKE-PSU variant has the least communication of any of the protocols we consider: up to an 8.8× improvement of Frikken, 125× improvement of DC, $10.9 - 13.6\times$

improvement of KRTW, and $3.9 - 5.7\times$ of GMRSS. It means that our scheme has great advantages in low bandwidth scenarios.

For our SKE-based protocol, the communication is dominated by t, i.e., the number of AND gates in the PRP decryption circuit. As mentioned in Section 5.2, we use LowMC [ARS+15] to minimize the number of AND gates, which is less than 800 for 128-bit security. Using LowMC, the communication of our SKE-PSU protocol is even lower than GMRSS.

6.2 Experimental Setup

We run our experiments on a single Intel Core i9-9900K with 3.3GHz and 128GB RAM. To better meet the potential deployment requirements, we do not use synchronous network connection. Instead, we use *Netty*[6] to maintain the communication channel. Netty is an asynchronous event-driven network application framework for the rapid development of maintainable high-performance protocols that are widely used for real applications. We design a unified data package format. Each data package contains a 256-bit length header and the actual payload bytes. The header is defined as follows (See Appendix G.1 for the detailed definition):

- Task ID (64-bit long): Each task is associated with a distinct task ID.
- Protocol ID (32-bit int): Each protocol is assigned with a distinct ID.
- Step ID (32-bit int): Each setup is also assigned with a distinct ID.
- Extra Info (64-bit long): Extra information for each step. For example, the current number of AND operations in SKE-2PC LowMC.
- Sender ID (32-bit int): Sender ID.
- Receiver ID (32-bit int): Receiver ID.
- Payload (bytes): the payload of the data package.

We use *Protocol Buffers*[7] for data serialization and deserialization. Protocol Buffers are Google's language-neutral, platform-neutral, extensible mechanism for serializing structured data and are fully compatible with Netty. Note that Protocol Buffers introduce lengths of each byte array in Payload Bytes in its serialization. Therefore, the actual communication costs are slightly higher than the theoretical communication costs. The results reported in our setting would reflect the actual costs when deploying protocols in real situations. We simulate the network connection using Linux `tc` command.

6.3 Implementation Details

Existing PSU implementations are under different MPC frameworks and different experimental settings. For example, the [KRTW19] implementation is under 128-bit element length while the [GMR+21] implementation is under 64-bit element length. Also, the [KRTW19] implementation supports multi-thread execution, while the [GMR+21] implementation does not. Further, the [GMR+21] implementation heavily relies on 1-out-of-2 OT, introducing recent silent OT technique may further reduce its communication cost [BCG+19, YWL+20]. However, existing efficient silent OT implementation [YWL+20] is available in *emp-toolkit*[8]. Combining these implementations relies on relatively heavy source code modification works.

After carefully studying existing open-source codes, we fully re-implement state-of-the-art PSU protocols [KRTW19, GMR+21] and their underlying basic protocols using Java, including base OT [NP01], OT extension [ALSZ13], silent OT [YWL+20], the specific OPRF variant [KKRT16], and GCT data structures. One may think that Java is much slower than C/C++. It is shown that although there is some performance gap, most basic operations in Java and C/C++ have similar performances[9]. For operations that have a huge efficiency gap between Java and C/C++, we use the Java Native Interface (JNI) technique to invoke C/C++ libraries. The details can be found in Appendix G.2. All experiments are under 64-bit element length.

[6] https://netty.io/

[7] https://developers.google.com/protocol-buffers

[8] https://github.com/emp-toolkit

[9] Our tests show that on Macbook Pro 2019, Java needs 0.095us for one AES operation, while C/C++ under AES instruction needs 0.071us. This is because Java would automatically use AES instruction if it detects that the current operating system supports it.

Our implementations support multi-thread executions for all PSU schemes, including [GMR+21], achieved by using Java Stream.parallel(). Our performance reports show that we obtain improved performance results for the [GMR+21] PSU scheme. Our complete implementation will be freely available on GitHub. We emphasize that designing a unified framework for all PSU protocols under C/C++ would further lead to better performance results, and we hope that our implementation can be a starting point.

6.4 Experimental Details

The SKE-PSU protocol is instantiated with the LowMC encryption scheme [ARS+15] where the block size and the key length are both 128 bits, and the number of Sboxes is $m = 10$ (i.e., the SboxLayer is a 10-folded parallel application of the basic 3-bit Sbox on the first 30 bits of the state, and for the remaining 88 bits, the SboxLayer is the identity). The concrete parameters in LowMC are from the Mobile PSI implementations provided by Kales et al. [KRS+19][10]. We use the improved inverse of the SBoxLayer provided by Liu et al. [LIM21] and follow the SBoxLayer implementation idea by Kales et al. [KRS+19] to implement the (non-2PC) decryption procedure. The underlying OKVSs for our PSU protocols are instantiated with our 3H-GCT++ in Fig. 3.

In SKE-PSU, we assume a commonly used setting where Boolean multiplication triples are pre-computed offline and stored locally. This follows real scenarios where Boolean multiplication triples are pre-generated by parties themselves or with the help of a Trusted-Third Party under the Trusted Dealer model.

In PKE-PSU, the ReRand-PKE is instantiated with the ECC ElGamal encryption scheme under the curve `secp256k1`. We found an interesting point in the implementation of PKE-PSU: In elliptic-curve-based cryptography, point compression is a standard trick, which can roughly reduce the representation of an EC point by half. The cost of this trick is that one has to perform point decompression in the future, which is typically considered to be cheap. Somewhat surprisingly, it turns out that point decompression is very costly. According to existing implementations provided in MCL and OpenSSL libraries, point decompression is as expensive as point exponentiation. Due to this fact, we prefer to use standard point representation for better efficiency when bandwidth is not of first priority. In the implementation, we use **PKE-PSU*** to represent the version that does not perform point compression.

We further introduce Silent OT [BCG+19, YWL+20] in the GMRSS scheme. We denote GMRSS schemes with Silent OT and standard OT extension by **GMRSS-Silent** and **GMRSS-IKNP**, respectively. The challenge is that current Silent OT implementations only provide Learning-Parity-with-Noise (LPN) parameters for large COT output sizes but not for small COT output sizes. For example, the Ferret OT [YWL+20] only provides LPN parameters that can output 10 million COTs. We follow a similar strategy introduced in [YWL+20] to find LPN parameters to output $2^{14}, 2^{16}, 2^{18}, 2^{20}, 2^{22}$ COTs in the regular-index setting, while all known attacks (e.g., Gaussian elimination, low-weight parity-check and information set decoding) require at least 2^{128} arithmetic operations. The parameters are shown in Table 3. We refer readers to see [YWL+20] for details on setting these parameters in Ferret OT.

The simulated network settings include typical LAN (10Gbps bandwidth and 0.02ms RTT latency) and WAN (10Mbps bandwidth and 80ms latency). We also introduce 100Mbps with 80ms latency and 1Gbps with 40ms latency as the intermediate network settings. In our KRTW implementation, we follow the pipelining optimization shown in [KRTW19] with 2^8 pipelining size when the receiver sends polynomials to the sender. In our PKE-PSU, we also leverage the pipelining optimization with the same 2^8 pipelining size when the sender sends ReRand outputs to the receiver.

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, base OT execution, and the one-time setup phase for Ferret OT [YWL+20]. The online phase does subsequent protocol executions. Note that in our PKE-PSU, the receiver can send the public key to the sender in the one-time setup phase, and all fixed-point precomputations related to the public key can also be done in that phase. We emphasize that fixed-point precomputations only need to be performed once, regardless of the number of subsequent protocol executions.

[10] https://github.com/contact-discovery/mobile_psi_cpp/blob/master/droidCrypto/lowmc/lowmc_128_128_20.c

# of COTs	One-time Setup			Iteration		
	k_0	n_0	t_0	k	n	t
2^{14}	1152	8792	581	1408	25167	1475
2^{16}	2304	12832	409	4352	78354	1411
2^{18}	2432	27451	872	15232	289584	1526
2^{20}	4864	71040	1131	55680	1119616	1536
2^{22}	12160	237343	1508	218880	4431616	1536
2^{24}	43776	882063	1533	860160	17658880	1536

Table 3: Extended Parameters in Ferret OT [YWL+20]

Detailed comparisons for set sizes $2^{14}, 2^{16}, 2^{18}, 2^{20}$ and controlled network configurations are shown in Table 4. We also show the relationship between communication/set size and set size of different schemes in Fig. 11. And Fig. 12 shows the relationship between the running time of different schemes and the bandwidth. Since the performance of GMRSS-Silent is worse than GMRSS-IKNP, we only compare the GMRSS-IKNP scheme in Fig. 12.

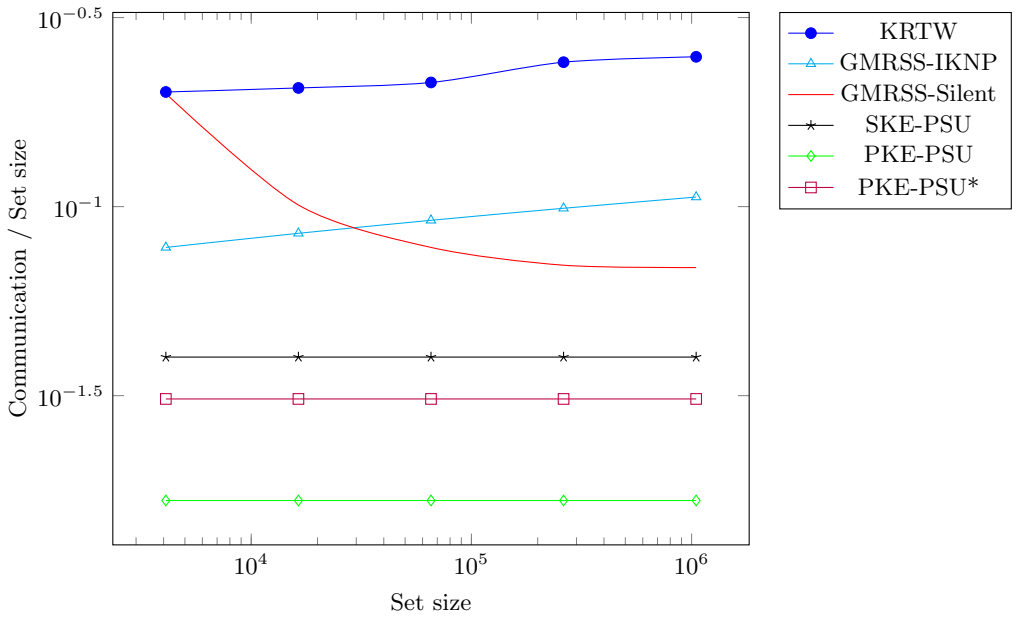

Fig. 11: Relationship between Communication / Set size and Set size for our protocols compared with KRTW,GMRSS-IKNP and GMRSS-Silent. Both x and y-axis is in log scale.

6.5 Performance Evaluation

Communication improvement. Our PKE-PSU protocol has the lowest communication among all protocols, which is $12.3-14.8\times$ lower than KRTW, $5.1-6.3\times$ lower than GMRSS-IKNP and $4.1-6.1\times$ lower than GMRSS-Silent. The communication of our SKE-PSU is about $2.5\times$ higher than that of PKE-PSU. The communication of our PKE-PSU* is about $2\times$ higher than that of PKE-PSU, which is due to the absence of point compression. Nevertheless, all our schemes have lower communication than that of KRTW and GMRSS schemes. As shown in Fig. 11, the communication costs of all our protocols are linear with the parties' set sizes, while the communication costs of the KRTW and GMRSS protocols are not. The larger the parties' set sizes are, the larger the communication cost

n	Protocol	Comm. (MB)					Running time (s)															
		$\mathcal{R}$		$\mathcal{S}$		total	LAN				1Gbps				100Mbps				10Mbps			
							$T=1$		$T=8$		$T=1$		$T=8$		$T=1$		$T=8$		$T=1$		$T=8$	
		setup	online	setup	online		setup	online	setup	online	setup	online	setup	online	setup	online	setup	online	setup	online	setup	online
2^{14}	KRTW	0.02	4.17	0.01	29.63	33.8	0.27	4.14	0.19	1.62	0.77	18.02	0.7	15.35	1.24	30.22	1.11	28.92	1.26	59.02	1.12	58.37
	GMRSS-IKNP	0.02	5.89	0.02	7.96	13.85	0.31	1.51	0.21	0.96	1.03	3.31	0.81	2.88	1.4	6.25	1.28	5.7	1.51	16.34	1.44	15.79
	GMRSS-Silent	0.17	1.86	0.17	14.76	16.62	2.13	2.57	1.98	1.72	3.42	7.26	3.27	6.84	4.49	12.84	4.32	11.76	4.67	25.07	4.64	24.1
	SKE-PSU	0.01	3.16	0	3.36	6.52	0.22	2.17	0.18	1.92	0.41	18.37	0.34	17.72	0.49	32.89	0.49	32.57	0.47	35.29	0.48	34.92
	PKE-PSU	0.01	1.16	0	1.59	2.75	4.76	3.14	4.73	1.91	5.05	3.68	5.01	2.49	5.25	4.86	5.22	3.78	5.32	6.25	5.32	5.61
	PKE-PSU*	0.01	2.16	0	2.9	5.05	4.75	2.68	4.68	1.34	5.03	3.3	4.99	2.22	5.31	4.55	5.28	3.67	5.35	7.89	5.27	7.16
2^{16}	KRTW	0.02	17.64	0.01	122.05	139.69	0.25	13.71	0.19	4.57	0.78	27.37	0.71	19.6	1.24	43.03	1.12	41.69	1.25	166.67	1.21	165.43
	GMRSS-IKNP	0.02	25.95	0.02	34.11	60.06	0.31	5.33	0.22	2.99	1.04	8.31	0.83	6.42	1.46	15.25	1.35	12.89	1.44	62.18	1.43	60.25
	GMRSS-Silent	0.2	6.7	0.2	44.44	51.15	2.12	9.66	2.01	5.71	3.43	14.69	3.35	11.76	4.46	24.04	4.3	21.1	4.81	63.52	4.56	61.37
	SKE-PSU	0.01	12.61	0	13.41	26.03	0.22	6.06	0.2	4.91	0.37	24.28	0.3	23.4	0.51	40.25	0.48	38.9	0.49	52.56	0.45	51.68
	PKE-PSU	0.01	4.62	0	6.37	10.99	4.76	10.9	4.71	5.61	5.05	11.98	4.99	6.76	5.26	13.27	5.26	8.15	5.22	18.65	5.29	16.24
	PKE-PSU*	0.01	8.63	0	11.57	20.19	4.71	9.56	4.72	3.81	5.12	10.57	5.1	4.9	5.27	12.33	5.29	6.96	5.3	24.08	5.29	22.09
2^{18}	KRTW	0.02	69.29	0.01	562.76	632.05	0.26	64.8	0.18	18.18	0.82	90.41	0.69	44.9	1.22	122.61	1.24	122.06	1.27	666.43	1.19	664.49
	GMRSS-IKNP	0.02	113.7	0.02	145.11	258.81	0.3	22.08	0.21	11.15	1.02	29.31	0.82	19.24	1.37	51.82	1.36	40.81	1.44	253.34	1.39	244.61
	GMRSS-Silent	0.29	26.15	0.29	156.79	182.93	2.29	38.01	2.17	21.33	3.66	47.63	3.82	31.64	4.81	66.7	5.1	51.43	5.35	217.16	5.33	201.13
	SKE-PSU	0.01	50.34	0	53.51	103.85	0.23	21.48	0.19	16.38	0.33	41.32	0.36	36.36	0.48	61.6	0.43	57.84	0.47	111.31	0.43	105.67
	PKE-PSU	0.01	18.5	0	25.45	43.95	4.73	43.4	4.72	21.39	5.05	44.78	4.99	23	5.2	47.91	5.23	26.01	5.24	69.24	5.27	58.62
	PKE-PSU*	0.01	34.5	0	46.26	80.76	4.71	37.51	4.7	13.85	5.1	39.5	5.01	15.8	5.32	44.56	5.27	21.06	5.29	87.97	5.29	81.59
2^{20}	KRTW	0.02	300.14	0.01	2305.8	2605.95	0.25	246.64	0.18	69.21	0.82	282.98	0.67	119.25	1.15	367.49	1.14	366.95	1.17	2647.86	1.2	2637.23
	GMRSS-IKNP	0.02	493.2	0.02	615.9	1109.1	0.27	100.59	0.19	52.09	0.86	122.08	0.81	78.89	1.46	210.25	1.34	160.06	1.49	1075.04	1.34	1034.18
	GMRSS-Silent	0.54	103.33	0.54	618.94	722.27	2.86	173.61	2.54	106.62	4.05	193.73	3.8	129.78	5.55	251.15	5.34	188.66	5.76	831.44	5.56	770.66
	SKE-PSU	0.01	200.88	0	213.55	414.43	0.24	85.24	0.19	63.7	0.31	107.95	0.29	82.87	0.5	144.62	0.43	118.37	0.48	390.67	0.43	368.39
	PKE-PSU	0.01	74	0	101.8	175.8	4.75	169.52	4.72	82.02	4.8	172.59	4.96	85.37	5.26	182.2	5.19	92.58	5.32	268.69	5.27	220.41
	PKE-PSU*	0.01	138	0	185	323	4.7	144.03	4.68	51.91	4.99	148.23	5.01	59.12	5.26	167.46	5.2	72.11	5.28	344.54	5.25	316.24

Table 4: Communication costs (in MB) and running time (in seconds) for our protocols compared with KRTW and GMRSS. The LAN network has 10 Gbps bandwidth and 0.2 ms RTT latency. All the other network settings have 40 ms RTT. Communication cost of $\mathcal{S}/\mathcal{R}$ indicates the outgoing communication from $\mathcal{S}/\mathcal{R}$ to the other party. The best protocol within a setting is marked in blue.

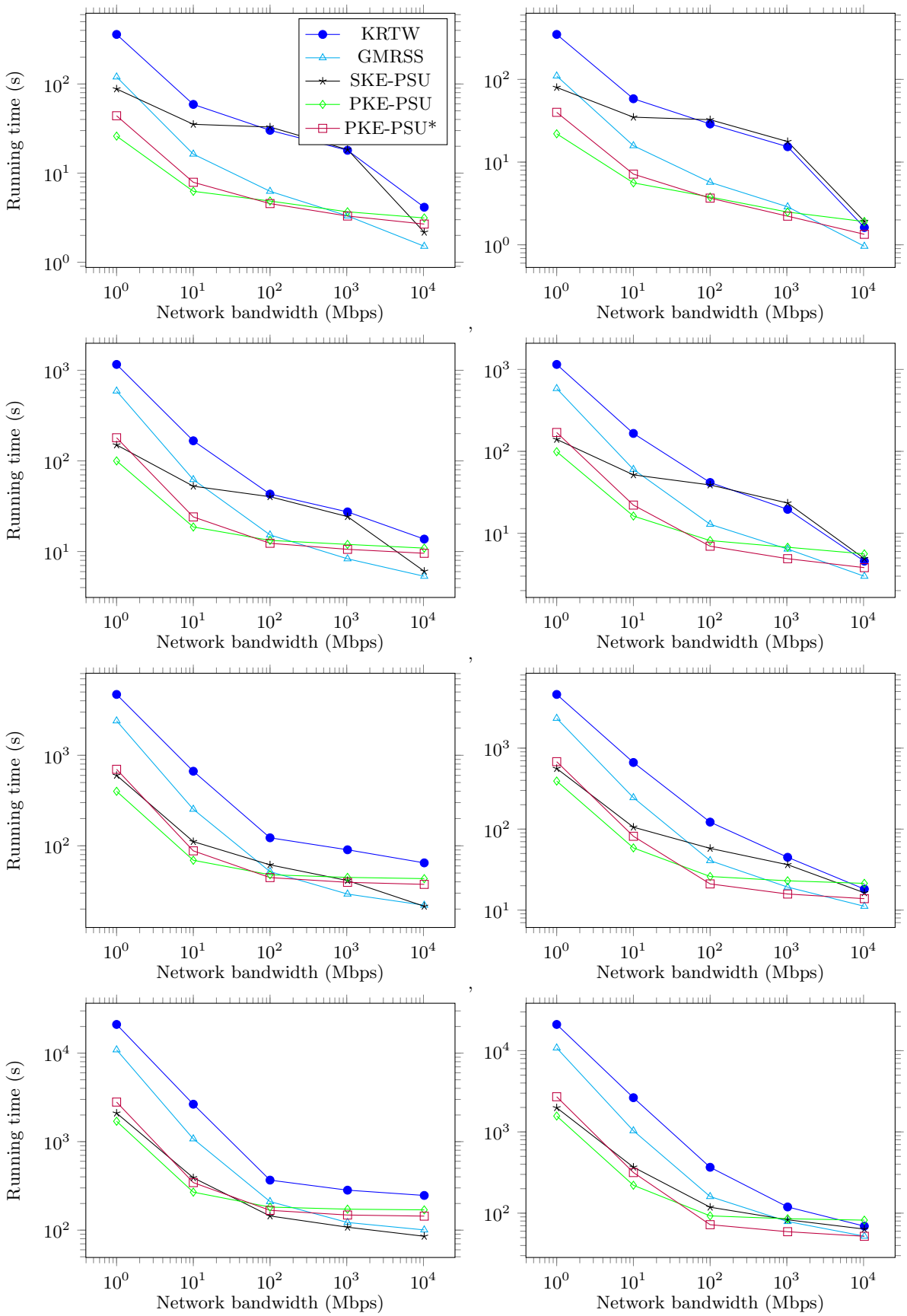

Fig. 12: Decline of running time (in seconds) on increasing network bandwidth for our protocols compared with KRTW and GMRSS. Both x and y-axis are in log scale. The four figures on the left correspond to $T = 1$ and the right correspond to $T = 8$. The corresponding set sizes from the first row to the last row are $n = 2^{14}, 2^{16}, 2^{18}$ and 2^{20} respectively

ratios are. We also note that the ratio of GMRSS-Silent decreased first and then increased, mainly due to the high overhead in the fixed initialization phase of Ferret OT.

Computation improvement. Our PKE-PSU scheme is the fastest one under low bandwidth, which is due to its lowest communication, e.g., for $n = 2^{20}$, PKE-PSU requires 220.41 seconds in 10Mbps with $T = 8$ threads, while KRTW requires 2637.23 seconds, a 12× improvement, GMRSS-IKNP requires 1034.18 seconds, a factor of 4.7× improvement, and GMRSS-Silent requires 770.66 seconds, a factor of 3.5× improvement. Our SKE-PSU performs best when the set size and the bandwidth are large. For example, for $n = 2^{20}$ with $T = 1$ thread in LAN setting, SKE-PSU requires 85.24 seconds, while KRTW requires 246.64 seconds, a 2.9× improvement, GMRSS-IKNP requires 100.59 seconds, a factor of 1.2× improvement, and GMRSS-Silent requires 173.61 seconds, a factor of 2× improvement. Our PKE-PSU* performs best in medium bandwidth (100Mbps and 1Gbps). For example, for $n = 2^{20}$ with $T = 8$ threads in 100Mbps, PKE-PSU* requires 72.11 seconds, while KRTW requires 366.95 seconds, a 5.1× improvement, GMRSS-IKNP requires 160.06 seconds, a factor of 2.2× improvement, and GMRSS-Silent requires 188.66 seconds, a factor of 2.6× improvement. We also noticed that the performance of PKE-PSU* improved significantly in the case of multithreading. For example, when increasing T from 1 to 8 in LAN setting, PKE-PSU* shows a factor of 2.8× improvement as the running time reduces from 144.03 seconds to 51.91 seconds for an input of $n = 2^{20}$ elements.

References

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

[ARS+15] Martin R. Albrecht, Christian Rechberger, Thomas Schneider, Tyge Tiessen, and Michael Zohner. Ciphers for MPC and FHE. In *EUROCRYPT 2015*, pages 430–454, 2015.

[BA12] Marina Blanton and Everaldo Aguiar. Private and oblivious set and multiset operations. In *ASIACCS 2012*, 2012.

[BCG+19] Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, Lisa Kohl, Peter Rindal, and Peter Scholl. Efficient two-round OT extension and silent non-interactive secure computation. In *CCS 2019*, 2019.

[Bea91] Donald Beaver. Efficient multiparty protocols using circuit randomization. In *CRYPTO 1991*, 1991.

[Bf12] M. Burkhart and Xenofontas Dimitropoulos fontas. Fast private set operations with sepia. 2012.

[BS05] Justin Brickell and Vitaly Shmatikov. Privacy-preserving graph algorithms in the semi-honest model. In *ASIACRYPT 2005*, 2005.

[Can01] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *FOCS 2001*, 2001.

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

[CPPT14] Ran Canetti, Omer Paneth, Dimitrios Papadopoulos, and Nikos Triandopoulos. Verifiable set operations over outsourced databases. In *PKC*, 2014.

[DC17] Alex Davidson and Carlos Cid. An efficient toolkit for computing private set operations. In *ACISP 2017*, 2017.

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

[FNO19] Brett Hemenway Falk, Daniel Noble, and Rafail Ostrovsky. Private set intersection with linear communication from general assumptions. In *WPES@CCS 2019*, 2019.

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

[Fri07] Keith B. Frikken. Privacy-preserving set union. In *ACNS 2007*, 2007.

[Gam85] Taher El Gamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Trans. Inf. Theory*, 31(4):469–472, 1985.

[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.

[GN19] Satrajit Ghosh and Tobias Nilges. An algebraic approach to maliciously secure private set intersection. In *EUROCRYPT 2019*, 2019.

[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.

[HKK+11] Jeongdae Hong, Jung Woo Kim, Jihye Kim, Kunsoo Park, and Jung Hee Cheon. Constant-round privacy preserving multiset union. Cryptology ePrint Archive, Report 2011/138, 2011. https://ia.cr/2011/138.

[HLS+16] Kyle Hogan, Noah Luther, Nabil Schear, Emily Shen, David Stott, Sophia Yakoubov, and Arkady Yerukhimovich. Secure multiparty computation for cooperative cyber risk assessment. In *SecDev 2016*, 2016.

[HN10] Carmit Hazay and Kobbi Nissim. Efficient set operations in the presence of malicious adversaries. In *PKC 2010*, 2010.

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

[KK13] Vladimir Kolesnikov and Ranjit Kumaresan. Improved OT extension for transferring short secrets. In *CRYPTO 2013*, 2013.

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

[KRS+19] Daniel Kales, Christian Rechberger, Thomas Schneider, Matthias Senker, and Christian Weinert. Mobile private contact discovery at scale. In *USENIX Security 2019*, 2019.

[KRTW19] Vladimir Kolesnikov, Mike Rosulek, Ni Trieu, and Xiao Wang. Scalable private set union from symmetric-key techniques. In *ASIACRYPT*, 2019.

[KS05] Lea Kissner and Dawn Xiaodong Song. Privacy-preserving set operations. In *CRYPTO 2005*, 2005.

[LIM21] Fukang Liu, Takanori Isobe, and Willi Meier. Cryptanalysis of full lowmc and lowmc-m with algebraic techniques. In *CRYPTO 2021*, 2021.

[LV04] Arjen K. Lenstra and Tim Voss. Information security risk assessment, aggregation, and mitigation. In *ACISP 2004*, 2004.

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

[NP01] Moni Naor and Benny Pinkas. Efficient oblivious transfer protocols. In *Proceedings of the Twelfth Annual Symposium on Discrete Algorithms*, 2001.

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

[PRTY20] Benny Pinkas, Mike Rosulek, Ni Trieu, and Avishay Yanai. PSI from paxos: Fast, malicious private set intersection. In *EUROCRYPT 2020*, 2020.

[PSSZ15] Benny Pinkas, Thomas Schneider, Gil Segev, and Michael Zohner. Phasing: Private set intersection using permutation-based hashing. In *USENIX Security 2015*, 2015.

[PSZ14] Benny Pinkas, Thomas Schneider, and Michael Zohner. Faster private set intersection based on OT extension. In *USENIX Security*, 2014.

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

[Rab05] Michael O. Rabin. How to exchange secrets with oblivious transfer. *IACR Cryptol. ePrint Arch.*, 2005:187, 2005.

[Reg05] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. In *STOC 2005*, 2005.

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

[SCK12] Jae Hong Seo, Jung Hee Cheon, and Jonathan Katz. Constant-round multi-party private set union using reversed laurent series. In *PKC 2012*, 2012.

[SM18] Katsunari Shishido and Atsuko Miyaji. Efficient and quasi-accurate multiparty private set union. In *SMARTCOMP 2018*, 2018.

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

[YWL+20] Kang Yang, Chenkai Weng, Xiao Lan, Jiang Zhang, and Xiao Wang. Ferret: Fast extension for correlated OT with small communication. In *CCS 2020*, 2020.

Appendix

A Encryption Schemes

A.1 Symmetric-key Encryption

A symmetric-key encryption (SKE) scheme is a tuple of four algorithms:

- $\mathsf{Setup}(1^\kappa)$: on input the security parameter κ outputs public parameters pp, which include the description of the message and ciphertext space M, C.
- $\mathsf{KeyGen}(pp)$: on input public parameters pp, outputs a key k.
- $\mathsf{Enc}(k, m)$: on input a key k and a plaintext $m \in M$, outputs a ciphertext $c \in C$.
- $\mathsf{Dec}(k, c)$: on input a key k and a ciphertext $c \in C$, outputs a message $m \in M$ or an error symbol $\perp$.

Correctness. For any $pp \leftarrow \mathsf{Setup}(1^\kappa)$, any $k \leftarrow \mathsf{KeyGen}(pp)$, any $m \in M$, and any $c \leftarrow \mathsf{Enc}(k, m)$, it holds that $\mathsf{Dec}(sk, c) = m$.
Security. For our purpose, we require a case-tailored security notion called *single-message multi-ciphertext pseudorandomness*. Formally, a SKE scheme is single-message multi-ciphertext pseudorandom if for any PPT $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$:

$$\mathsf{Adv}_{\mathcal{A}}(1^\kappa) = \Pr\left[\beta = \beta' : \begin{array}{l} pp \leftarrow \mathsf{Setup}(1^\kappa); \\ k \leftarrow \mathsf{KeyGen}(pp); \\ (m, state) \leftarrow \mathcal{A}_1(pp); \\ \beta \xleftarrow{\mathrm{R}} \{0,1\}; \\ c^*_{i,0} \leftarrow \mathsf{Enc}(k, m), c^*_{i,1} \xleftarrow{\mathrm{R}} C, \text{ for } i \in [n]; \\ \beta' \leftarrow \mathcal{A}_2(pp, state, \{c^*_{i,\beta}\}_{i\in[n]}) \end{array}\right] - \frac{1}{2}$$

is negligible in κ.

Remark 1. The single-message multi-ciphertext pseudorandomness is a mild security notion that is satisfied by most IND-CPA secure SKE schemes, for instance, the classical PRF-based SKE.

A.2 Re-randomizable PKE

A re-randomizable PKE (ReRand-PKE) scheme is a tuple of five algorithms:

- $\mathsf{Setup}(1^\kappa)$: on input the security parameter κ outputs public parameters pp, which include the description of the message and ciphertext space M, C.
- $\mathsf{KeyGen}(pp)$: on input public parameter pp, outputs a keypair (pk, sk).
- $\mathsf{Enc}(pk, m)$: on input a public key pk and a message $m \in M$, outputs a ciphertext $c \in C$.
- $\mathsf{Dec}(sk, c)$: on input a secret key sk and a ciphertext $c \in C$, outputs a message $m \in M$ or an error symbol $\perp$.
- $\mathsf{ReRand}(pk, c)$: on input a public key pk and a ciphertext $c \in C$, outputs another ciphertext $c' \in C$.

Correctness. For any $pp \leftarrow \mathsf{Setup}(1^\kappa)$, any $(pk, sk) \leftarrow \mathsf{KeyGen}(pp)$, any $m \in M$, any $c \leftarrow \mathsf{Enc}(pk, m)$, and any $c' \leftarrow \mathsf{ReRand}(pk, c)$, it holds that $\mathsf{Dec}(sk, c) = \mathsf{Dec}(sk, c') = m$.
Indistinguishability. For any $pp \leftarrow \mathsf{Setup}(1^\kappa)$, any $(pk, sk) \leftarrow \mathsf{KeyGen}(pp)$, and any $m \in M$, the distribution $c_0 \leftarrow \mathsf{Enc}(pk, m)$ and the distribution $c_1 \leftarrow \mathsf{ReRand}(pk, c_0)$ are identical.
Security. For our purpose, we require a case-tailored security notion called *single-message multi-ciphertext pseudorandomness*. Formally, a PKE scheme is single-message multi-ciphertext pseudorandom if for any PPT $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$:

$$\mathsf{Adv}_{\mathcal{A}}(1^\kappa) = \Pr\left[\beta = \beta' : \begin{array}{l} pp \leftarrow \mathsf{Setup}(1^\kappa); \\ (pk, sk) \leftarrow \mathsf{KeyGen}(pp); \\ (m, state) \leftarrow \mathcal{A}_1(pp, pk); \\ \beta \xleftarrow{\mathrm{R}} \{0,1\}; \\ c^*_{i,0} \leftarrow \mathsf{Enc}(pk, m), c^*_{i,1} \xleftarrow{\mathrm{R}} C, \text{ for } i \in [n]; \\ \beta' \leftarrow \mathcal{A}_2(pp, state, \{c^*_{i,\beta}\}_{i\in[n]}) \end{array}\right] - \frac{1}{2}$$

is negligible in κ.

Remark 2. We remark that single-plaintext multi-ciphertext pseudorandomness is a very mild property for PKE. This is because most natural IND-CPA secure PKE constructions satisfy single-message single-ciphertext pseudorandomness, which further implies single-plaintext multi-ciphertexts pseudorandomness via a standard hybrid argument.

It is straightforward to verify that the DDH-based ElGamal PKE [Gam85] and Regev's LWE-based PKE [Reg05] are re-randomizable PKE schemes satisfying the above correctness, indistinguishability, and single-plaintext multi-ciphertext pseudorandomness.

B Oblivious Key-Value Store Scheme

B.1 Instantiation of OKVSs

We recall some instantiations of OKVS and analyze their parameters.

Polynomial. Polynomial can be seen as a natural OKVS: to insert n key-value pairs $\{(x_i, y_i)\}_{i\in[n]}$, one computes P as the polynomial which passes through points $\{(x_i, y_i)\}_{i\in[n]}$.

The advantage of polynomial is that its rate reaches optimal 1, which induces the lowest communication in the protocol. However, its encoding and decoding are less efficient. Using the optimization of [PRTY19], the encoding and decoding complexity are respectively $O(n \log^2 n)$ and $O(\log n)$. Another disadvantage of polynomial is that it only satisfies correctness and obliviousness, not randomness because polynomial generation is a deterministic algorithm.

Garbled Bloom Filter. Garbled Bloom Filter (GBF) was introduced in [DCW13] in the context of PSI protocols. The values are taken from $\mathbb{F}_{2^\sigma}$. A GBF is an m-long array D associated with k random functions $h_1, \ldots, h_k : \{0,1\}^* \to [m]$. Let $D[j]$ denote the jth component of array D. To insert a key-value pair (x, y) in a GBF, one chooses random $D[h_i(x)]$ for $i \in [k]$ conditioned on $y = \oplus_{i\in[k]} D[h_i(x)]$.

In [DCW13], they showed that if the GBF has size $m = O(\lambda n)$ then the generation of GBF succeeds with probability $1 - 2^{-\lambda}$. Therefore, the rate of GBF is $O(1/\lambda)$. The encoding complexity is $O(\lambda n)$ and decoding requires λ XOR at most.

Garbled Cuckoo Table. Garbled Cuckoo Table (GCT) was introduced in [PRTY20] as an optimization of GBF. The idea of GCT is similar to GBF, and the difference is that GCT uses only two hash functions instead of λ. However, two hash functions will cause a non-negligible probability of failure. To solve this problem, they introduce some additional positions and use a new random function to map the key to these positions. They use cuckoo graph to analyze the probability of success, and finally they achieve a better rate, which is about 0.42.

However, as Rindal and Schoppmann [RS21] pointed out, the original GCT scheme [PRTY20] does not meet the obliviousness properties we defined before. The main reason is that the original GCT needs to solve a linear equation to satisfy the key-value constraint. However, the free variable in the equation is set to zero, which means keys are no longer randomly shared in some positions like GBF. As a result, the GCT has some zeros depending on the key's set. They made a little modification to make GCT (they called XoPaXoS) meet this property. The main idea is to first assign random values to the free variables, and then solve the remaining full rank equations. Recently, Garimella et al. [GPR+21] improved original GCT to 3H-GCT, the rate is increased to 0.81 by using three hash functions. However, the original 3H-GCT still assigns zero to the free variables in linear equation. We use similar modifications to make 3H-GCT meet Obliviousness and Randomness, as we described in Fig. 3.

We summarize the parameters and properties of the above schemes in Table 5.

B.2 Property Proof

The correctness is obvious. Now we prove the Obliviousness and Randomness of our 3H-GCT++.

Theorem 7. *3H-GCT++ in Fig. 3 satisfies the Obliviousness and Randomness.*

Proof. **Obliviousness**: As we described before, 3H-GCT++ is generated by additive secret sharing of values at the random position mapped by hash function, and selecting random value at the point not

Table 5: A comparison between the different OKVS schemes.

scheme	rate	encoding	decoding	obliviousness	randomness
Polynomial	1	$O(n \log^2 n)$	$O(\log n)$	$\surd$	$\times$
GBF [DCW13]	$O(1/\lambda)$	$O(\lambda n)$	$O(\lambda)$	$\surd$	$\surd$
2H-GCT [PRTY20]	$0.42 - o(1)$	$O(\lambda n)$	$O(\lambda)$	$\times$	$\times$
XoPaXoS [RS21]	$0.42 - o(1)$	$O(\lambda n)$	$O(\lambda)$	$\surd$	$\surd$
3H-GCT [GPR+21]	$0.81 - o(1)$	$O(\lambda n)$	$O(\lambda)$	$\times$	$\times$
3H-GCT++ in Fig. 3	$0.81 - o(1)$	$O(\lambda n)$	$O(\lambda)$	$\surd$	$\surd$

n is the number of key-value pairs, λ is a statistical security parameter (e.g.,$\lambda = 40$).

mapped. Since the value are uniform distribution, we have that $\{(D_1, \ldots, D_m) | D_i \leftarrow \mathbb{G}, i \in [m]\} \equiv \{(D_1, \ldots, D_m) | y \leftarrow \mathbb{G}, D_i \leftarrow \mathbb{G}, i \in [m-1], D_m := y - \sum_{i \in [m-1]} D_i\}$, which implies Obliviousness.

Randomness: Let $X = \{x_1, \ldots, x_n\}$ denote the key's set. For any $x^* \notin X$, let $\mathrm{row}(x^*)$ defined as before. There are three cases:

Case 1: $\exists x_i \in X$ such that $\mathrm{row}(x_i) = \mathrm{row}(x^*)$. By the parameter of GCT scheme [PRTY20, GPR+21], this probability is $2^{-\lambda}$.

Case 2: Let $o(x_i) \subset [m]$ be the set of positions that are 1s of $row(x_i)$, $i \in [n]$ and let $O := \cup_{i \in [n]} o(x_i)$. In this case, $o(x^*) \subset O$, that is, all the 1 positions of $\mathrm{row}(x^*)$ have been mapped when generating D. Now we can divide $o(x^*)$ into several groups according to which key is mapped to that location. If there is a location i mapped by both different keys, then the location i can be randomly put into one of the groups. Since the different positions of a key mapped to corresponds to an additive secret sharing of the corresponding value, the sum of each group should be a uniformly random element in $\mathbb{G}$. Therefore $\mathsf{Decode}_H(D, x^*)$ is a uniformly random string.

Case 3: $\exists j \in [m]$ such that $j \in o(x^*) \wedge j \notin O$, that is, there are some positions of $\mathrm{row}(x^*)$ were not mapped when generating D. By the generation of GCT, those positions not mapped are assigned with a random value. Therefore $\mathsf{Decode}_H(D, x^*)$ is a uniformly random string.

In summary, with probability $1 - 2^{-\lambda}$, $\mathsf{Decode}_H(D, x^*)$ is a uniformly random string.

C Proof of Theorem 1

Below we give the details of the proof of Theorem 1.

Proof. We exhibit simulators $\mathsf{Sim}_{\mathcal{R}}$ and $\mathsf{Sim}_{\mathcal{S}}$ for simulating corrupt $\mathcal{R}$ and $\mathcal{S}$ respectively, and argue the indistinguishability of the produced transcript from the real execution.

Corrupt sender: $\mathsf{Sim}_{\mathcal{S}}(Y = \{y_1, \ldots, y_{n_\mathsf{y}}\})$ simulates the view of corrupt semi-honest sender. It executes as follows:

1. $\mathsf{Sim}_{\mathcal{S}}$ selects n_x random key-value pairs $(x_i, s_i)_{i \in [n_\mathsf{x}]}$, where x_i and s_i are random item and ciphertext respectively. Then $\mathsf{Sim}_{\mathcal{S}}$ computes OKVS $D := \mathsf{Encode}_H((h(x_1), s_1), \ldots, (h(x_{n_\mathsf{x}}), s_{n_\mathsf{x}}))$ and appends it to the view.
2. $\mathsf{Sim}_{\mathcal{S}}$ computes $s_i^* = \mathsf{Decode}_H(D, h(y_i))$ for $i \in [n_\mathsf{y}]$. Then, it invokes VODM simulator $\mathsf{Sim}^S_{\mathsf{vodm}}(s_1^*, \ldots, s_{n_\mathsf{y}}^*)$ 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. We formally prove this by a standard hybrid argument method. We define four hybrid transcripts T_0, T_1, T_2, T_3 where T_0 is real view of $\mathcal{S}$, and T_3 is the output of $\mathsf{Sim}_{\mathcal{S}}$.

- Hybrid_0. The first hybrid is the real interaction described in Fig. 7. Here, an honest $\mathcal{R}$ uses input X, honestly interacts with the corrupt $\mathcal{S}$. Let T_0 denote the real view of $\mathcal{S}$.
- Hybrid_1. Let T_1 be the same as T_0, except that $(s_1, \ldots, s_{n_\mathsf{x}})$ are replaced by n_x random ciphertexts. This hybrid is computationally indistinguishable from T_0 by the single-message multi-ciphertext pseudorandomness of the encryption scheme.
  Specifically, if there is a distinguisher $\mathcal{D}$ can distinguish T_0 and T_1 with non-negligible probability, then we can construct a PPT adversary $\mathcal{A}$ to break the single-message multi-ciphertext pseudorandomness of encryption scheme. $\mathcal{A}$ works as follows: when $\mathcal{A}$ receives pp from challenger, $\mathcal{A}$ selects

a random s as challenge message. Then $\mathcal{A}$ receives ciphertexts $\{c_i^*\}_{i\in[n_\times]}$ from challenger. Now $\mathcal{A}$ executes as an honest receiver with the corrupt $\mathcal{S}$ except step 2. In this step, $\mathcal{A}$ computes OKVS as $D := \mathsf{Encode}_H(\{h(x_i), c_i^*\}_{i\in[n_\times]})$. Now $\mathcal{A}$ invokes $\mathcal{D}$ with the sender's view in the above interaction and outputs $\mathcal{D}$'s output. Note that if $\{c_i^*\}_{i\in[n_\times]}$ are the encryption of s, the view of corrupt sender is exactly the real view, which corresponds to T_0. If $\{c_i^*\}_{i\in[n_\times]}$ are random ciphertexts, the view corresponds to T_1. Therefore, $\mathcal{A}$ can break the security of the encryption scheme with the same advantages as $\mathcal{D}$.

- Hybrid$_2$. Let T_2 be the same as T_1, except that the inputs of the receiver $\mathcal{R}$ are replaced by $n_\times$ random items. Note that the selection of value in OKVS has been replaced with random ciphertexts in T_1. By the obliviousness property of OKVS, T_1 and T_2 are statistically indistinguishable.
- Hybrid$_3$. Let T_3 be the same as T_2, except that the VODM execution is replaced by simulator $\mathsf{Sim}^S_{\mathsf{vodm}}$. The security of VODM functionality guarantees the view is indistinguishable from real execution.

Corrupt receiver: $\mathsf{Sim}_{\mathcal{R}}(X = \{x_1, \ldots, x_{n_\times}\}, b)$ simulates the view of corrupt semi-honest receiver. It executes as follows:

1. $\mathsf{Sim}_{\mathcal{R}}$ selects a random $s \leftarrow \{0,1\}^\sigma$ and generates a random encryption key k as the semi-honest receiver does in the real protocol. Then, it invokes VODM simulator $\mathsf{Sim}^R_{\mathsf{vodm}}(k, s, b)$ and appends the output to the view.

The view output by $\mathsf{Sim}_{\mathcal{R}}$ is indistinguishable from the real one by the underlying simulators' indistinguishability.

D SKE-based Multi-Query RPMT

Now we show that PRF-based SKE satisfies the single-message multi-ciphertext pseudorandomness property.

Let $F := \{f_k : \{0,1\}^\kappa \to \{0,1\}^\kappa\}_{k\in K}$ be a PRF family. The PRF-based SKE scheme is as follows:

- $\mathsf{Setup}(1^\kappa)$: on input the security parameter κ outputs public parameters pp, which include the description of the message and ciphertext space $M = \{0,1\}^\kappa, C = \{0,1\}^{2\kappa}$.
- $\mathsf{KeyGen}(pp)$: on input public parameter pp, outputs a key $k \xleftarrow{\text{R}} K$.
- $\mathsf{Enc}(k, m)$: on input a key k and a plaintext $m \in M$, chooses $r \xleftarrow{\text{R}} \{0,1\}^\kappa$, outputs a ciphertext $c = (r, f_k(r) \oplus m)$.
- $\mathsf{Dec}(k, c)$: on input a key k and a ciphertext $c = (r, c_2)$, outputs $m = f_k(r) \oplus c_2$.

Next, we prove the single-message multi-ciphertext pseudorandomness of the above PRF-based SKE scheme.

Lemma 1. *The PRF-based SKE satisfies the single-message multi-ciphertext pseudorandomness property defined in Section A.1.*

Proof. If there is a PPT adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ can break the single-message multi-ciphertext pseudorandomness of SKE scheme, then we can construct a PPT adversary $\mathcal{B}$ to break the security of PRF. In particular, $\mathcal{B}$ runs Setup to obtain pp, then it invokes $\mathcal{A}_1(pp)$ to obtain $(m, state)$. Now, $\mathcal{B}$ selects $r_i \xleftarrow{\text{R}} \{0,1\}^\kappa$ and queries the oracle with r_i to obtain $f(r_i)$ for $i \in [n]$. Then, $\mathcal{B}$ sets $c_i = (r_i, f(r_i) \oplus m)$ and invokes $\mathcal{A}_2(pp, state, \{c_i\}_{i\in[n]})$ to obtain a bit β'. Finally, $\mathcal{B}$ outputs β'.

If f is PRF: $\{c_i\}_{i\in[n]}$ are exactly the n times encryption of m, which correspond to $\beta = 0$.

If f is random function: $f(r)$ is also a random string on $\{0,1\}^\kappa$, which means c_i is a random distribution in ciphertext space, corresponding to $\beta = 1$.Therefore, $\mathcal{B}$ distinguishes PRF with the same probability as $\mathcal{A}$ in single-message multi-ciphertext pseudorandomness experiment.

E PSU Construction from Leaky VODM

We give the PSU protocol from leaky VODM in Fig. 13.

Parameters:

- Two parties: sender $\mathcal{S}$ and receiver $\mathcal{R}$.
- A ReRand-PKE scheme $(\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{ReRand})$.
- An OKVS scheme $(\mathsf{Encode}_H, \mathsf{Decode}_H)$.
- A collision-resistant hash function $h(x) : \{0,1\}^* \to \{0,1\}^\sigma$.

Input of $\mathcal{S}$: $Y = \{y_1, \ldots, y_{n_\mathsf{y}}\} \subset \{0,1\}^*$
Input of $\mathcal{R}$: $X = \{x_1, \ldots, x_{n_\mathsf{x}}\} \subset \{0,1\}^*$
Protocol:

1. $\mathcal{R}$ selects a random *indication string* $s \in \mathbb{F}_{2^\sigma}$. $\mathcal{R}$ also generates a random key pair $pp \leftarrow \mathsf{Setup}$, $(pk, sk) \leftarrow \mathsf{KeyGen}(pp)$, a randomness set $R = \{r_1, \ldots, r_{n_\mathsf{x}}\}$ and computes $s_i := \mathsf{Enc}(pk, s; r_i)$ for $i \in [n_\mathsf{x}]$.
2. $\mathcal{R}$ computes an OKVS $D := \mathsf{Encode}_H((h(x_1), s_1), \ldots, (h(x_{n_\mathsf{x}}), s_{n_\mathsf{x}}))$.
3. $\mathcal{R}$ sends D and pk to $\mathcal{S}$.
4. $\mathcal{S}$ computes $s_i^* := \mathsf{Decode}_H(D, h(y_i))$ for $i \in [n_\mathsf{y}]$.
5. $\mathcal{S}$ and $\mathcal{R}$ invoke the leaky VODM functionality $\mathcal{F}_{\mathsf{lvodm}}$. The sender $\mathcal{S}$ acts as sender in leaky VODM with input $\{s_i^*\}_{i \in [n_\mathsf{y}]}$ and receives nothing. The receiver $\mathcal{R}$ acts as receiver in leaky VODM with input (pk, sk, s) and receives $b \in \{0,1\}^{n_\mathsf{y}}$ and $\{s_i' | b_i = 0\}$.
6. $\mathcal{R}$ initialize set $Z := \{\}$.
7. For $i \in [n_\mathsf{y}]$:
   (a) $\mathcal{S}$ and $\mathcal{R}$ invoke the OT functionality $\mathcal{F}_{\mathsf{ot}}$
   (b) $\mathcal{S}$ acts as sender with input $(y_i, \perp)$.
   (c) $\mathcal{R}$ acts as receiver with input b_i.
   (d) $\mathcal{R}$ obtains the OT output z_i and sets $Z = Z \cup \{z_i\}$
8. $\mathcal{R}$ outputs $X \cup Z$.

Fig. 13: PSU from leaky VODM Π_{psu}

Theorem 8. *Assume the Re-Rand PKE scheme $\mathcal{E} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ satisfies single-message multi-ciphertext pseudorandomness. The protocol in Fig. 13 securely computes $\mathcal{F}_{\mathsf{psu}}$ against semi-honest adversaries in the $(\mathcal{F}_{\mathsf{lvodm}}, \mathcal{F}_{\mathsf{ot}})$-hybrid model.*

Proof. We exhibit simulators $\mathsf{Sim}_\mathcal{R}$ and $\mathsf{Sim}_\mathcal{S}$ for simulating corrupt $\mathcal{R}$ and $\mathcal{S}$ respectively, and argue the indistinguishability of the produced transcript from the real execution.

Corrupt Sender: $\mathsf{Sim}_\mathcal{S}(Y = \{y_1, \ldots, y_{n_\mathsf{y}}\})$ simulates the view of corrupt semi-honest sender. It executes as follows:

1. $\mathsf{Sim}_\mathcal{S}$ selects n_x random key-value pairs $(x_i, s_i)_{i \in [n_\mathsf{x}]}$, where x_i and s_i are random item and ciphertext respectively. Then $\mathsf{Sim}_\mathcal{S}$ generates $pp \leftarrow \mathsf{Setup}$, $(pk, sk) \leftarrow \mathsf{KeyGen}(pp)$, computes $D = \mathsf{Encode}_H((h(x_1), s_1), \ldots, (h(x_{n_\mathsf{x}}), s_{n_\mathsf{x}}))$ and appends (pk, D) to the view.
2. $\mathsf{Sim}_\mathcal{S}$ computes $s_i^* = \mathsf{Decode}_H(D, h(y_i))$ for $i \in [n_\mathsf{y}]$. Then, it invokes leaky VODM simulator $\mathsf{Sim}^S_{\mathsf{lvodm}}(s_1^*, \ldots, s_{n_\mathsf{y}}^*)$ and appends the output to the view.
3. For $i \in [n_\mathsf{y}]$, $\mathsf{Sim}_\mathcal{S}$ invokes OT simulator $\mathsf{Sim}^S_{\mathsf{ot}}(y_i, \perp)$ 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. We formally prove this by a standard hybrid argument method. We define four hybrid transcripts T_0, T_1, T_2, T_3 where T_0 is real view of $\mathcal{S}$, and T_3 is the output of $\mathsf{Sim}_\mathcal{S}$.

- Hybrid_0. The first hybrid is the real interaction described in Fig. 13. Here, an honest $\mathcal{R}$ uses input X, honestly interacts with the corrupt $\mathcal{S}$. Let T_0 denote the real view of $\mathcal{S}$.
- Hybrid_1. Let T_1 be the same as T_0, except that $(s_1, \ldots, s_{n_\mathsf{x}})$ are replaced by n_x random ciphertexts. This hybrid is computationally indistinguishable from T_0 by the single-message multi-ciphertext pseudorandomness of the encryption scheme.
  Specifically, if there is a distinguisher $\mathcal{D}$ can distinguish T_0 and T_1 with non-negligible probability, then we can construct a PPT adversary $\mathcal{A}$ to break the single-message multi-ciphertext pseudorandomness of encryption scheme. $\mathcal{A}$ works as follows: when $\mathcal{A}$ receives pp from challenger, $\mathcal{A}$ selects

a random s as challenge message. Then $\mathcal{A}$ receives ciphertexts $\{c_i^*\}_{i\in[n_\mathsf{x}]}$ from challenger. Now $\mathcal{A}$ executes as an honest receiver with the corrupt $\mathcal{S}$ except step 2. In this step, $\mathcal{A}$ computes OKVS as $D := \mathsf{Encode}_H(\{h(x_i), c_i^*\}_{i\in[n_\mathsf{x}]})$. Now $\mathcal{A}$ invokes $\mathcal{D}$ with the sender's view in the above interaction and outputs $\mathcal{D}$'s output. Note that if $\{c_i^*\}_{i\in[n_\mathsf{x}]}$ are the encryption of s, the view of corrupt sender is exactly the real view, which corresponds to T_0. If $\{c_i^*\}_{i\in[n_\mathsf{x}]}$ are random ciphertexts, the view corresponds to T_1. Therefore, $\mathcal{A}$ can break the security of the encryption scheme with the same advantages as $\mathcal{D}$.

- Hybrid$_2$. Let T_2 be the same as T_1, except that the inputs of the receiver $\mathcal{R}$ are replaced by n_x random items. Note that the selection of value in OKVS has been replaced with random ciphertexts in T_1. By the obliviousness property of OKVS, T_1 and T_2 are statistically indistinguishable.
- Hybrid$_3$. Let T_3 be the same as T_2, except that the leaky VODM and OT execution is replaced by simulator $\mathsf{Sim}^S_{\mathsf{lvodm}}$ and $\mathsf{Sim}^S_{\mathsf{ot}}$. The security of leaky VODM and OT functionality guarantee the view is indistinguishable from real execution.

Corrupt Receiver: $\mathsf{Sim}_\mathcal{R}(X = \{x_1, \ldots, x_{n_\mathsf{x}}\}, X \cup Y)$ simulates the view of corrupt receiver. It executes as follows:

1. $\mathsf{Sim}_\mathcal{R}$ executes first two step as an honest receiver and obtains $s, (pk, sk), D$.
2. $\mathsf{Sim}_\mathcal{R}$ define the set $Z := X \cup Y \setminus X$, i.e. the set of elements that Y "brings to the union". Next, it uses $\perp$ to pads Z to n_y elements and permutates these elements randomly. Let $Z = \{z_1, \ldots, z_{n_\mathsf{y}}\}$.
3. $\mathsf{Sim}_\mathcal{R}$ sets $b_i = 1$ if and only if $z_i \in X$ for $i \in [n_\mathsf{y}]$. For $z_i \notin X$, $\mathsf{Sim}_\mathcal{R}$ computes $s'_i := \mathsf{Dec}(sk, \mathsf{Decode}_H(D, h(z_i)))$ Then, it invokes leaky vectro ODM simulator $\mathsf{Sim}^R_{\mathsf{lvodm}}(s, b, \{s'_i | b_i = 0\})$ and appends the output to the view.
4. For $i \in [n_\mathsf{y}]$, $\mathsf{Sim}_\mathcal{R}$ invokes OT simulator $\mathsf{Sim}^R_{\mathsf{ot}}(b_i, z_i)$ and appends the output to the view.

Now we argue that the view output by $\mathsf{Sim}_\mathcal{R}$ is indistinguishable from the real one. In the simulation, the way $\mathcal{R}$ obtains the elements in $Z = X \setminus Y$ is identical to the real execution. By the underlying simulators' indistinguishability, the simulated view is computationally indistinguishable from the real one.

F Multi-Query RPMT Based on Membership Encryption

We describe how to construct mq-RPMT using the language of Membership Encryption (ME). As we mentioned in Section 4.3, this will help us reduce the communication by half when sending OKVS. We first define the vector oblivious decryption (VOD) functionality in Fig. 14.

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

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

Fig. 14: Vector Oblivious Decryption Functionality $\mathcal{F}_{\mathsf{vod}}$

Now, we use the language of ME to describe how to construct mq-RPMT. The formal protocol is described in Fig. 15.

Correctness. For all $i \in [n_\mathsf{y}]$, if $y_i \in X$, there is a $x_j \in X, j \in [n_\mathsf{x}]$ s.t. $y_i = x_j$. In this case, $s_i^* = \mathsf{Decode}_H(D, h(x_j)) = s_j$. Since $s_j = \mathsf{Enc}(k, x_j)$, we have $\mathsf{Dec}(k, s_j) = 1$. In the case $y_i \notin X$, if hash functions collide, that is, $h(y_i) = h(x)$ for some $y_i \notin X$, the correctness will be violated. By setting $\sigma = \lambda + \log n_\mathsf{x} n_\mathsf{y}$, a union bound shows probability of collision is negligible $2^{-\lambda}$. When no

collision occurs, from the randomness of OKVS, $s_i^* = \mathsf{Decode}_H(D, h(y_i))$ is a random ciphertext, result in $\mathsf{Dec}(k, s_i^*) = 0$ with overwhelming probability. The union bound guarantees that for all $y_i \notin X$, the probability that there exists an s_i^* s.t. $\mathsf{Dec}(k, s_i^*) = 1$ is negligible.

We now state and prove the security properties of the above mq-RPMT protocol.

Theorem 9. *Assume $\mathcal{E} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ is a membership encryption scheme as we defined in section 4.3. The protocol in Fig. 15 securely computes $\mathcal{F}_{\mathsf{mq\text{-}rpmt}}$ against semi-honest adversaries in the $\mathcal{F}_{\mathsf{vod}}$-hybrid model.*

Proof. We exhibit simulators $\mathsf{Sim}_{\mathcal{R}}$ and $\mathsf{Sim}_{\mathcal{S}}$ for simulating corrupt $\mathcal{R}$ and $\mathcal{S}$ respectively, and argue the indistinguishability of the produced transcript from the real execution.

Corrupt sender: $\mathsf{Sim}_{\mathcal{S}}(Y = \{y_1, \ldots, y_{n_\mathsf{y}}\})$ simulates the view of corrupt semi-honest sender. It executes as follows:

1. $\mathsf{Sim}_{\mathcal{S}}$ selects n_x random key-value pairs $(x_i, s_i)_{i \in [n_\mathsf{x}]}$, where x_i and s_i are random item and ciphertext respectively. Then $\mathsf{Sim}_{\mathcal{S}}$ computes OKVS $D := \mathsf{Encode}_H((h(x_1), s_1), \ldots, (h(x_{n_\mathsf{x}}), s_{n_\mathsf{x}}))$ and appends it to the view.
2. $\mathsf{Sim}_{\mathcal{S}}$ computes $s_i^* = \mathsf{Decode}_H(D, h(y_i))$ for $i \in [n_\mathsf{y}]$. Then, it invokes VOD simulator $\mathsf{Sim}_{\mathsf{vod}}^S(s_1^*, \ldots, s_{n_\mathsf{y}}^*)$ 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. We formally prove this by a standard hybrid argument method. We define four hybrid transcripts T_0, T_1, T_2, T_3 where T_0 is real view of $\mathcal{S}$, and T_3 is the output of $\mathsf{Sim}_{\mathcal{S}}$.

- Hybrid$_0$. The first hybrid is the real interaction described in Fig. 15. Here, an honest $\mathcal{R}$ uses input X, honestly interacts with the corrupt $\mathcal{S}$. Let T_0 denote the real view of $\mathcal{S}$.
- Hybrid$_1$. Let T_1 be the same as T_0, except that $(s_1, \ldots, s_{n_\mathsf{x}})$ are replaced by n_x random ciphertexts. This hybrid is computationally indistinguishable from T_0 by the multi-elements pseudorandomness of the membership encryption scheme.
  Specifically, if there is a distinguisher $\mathcal{D}$ can distinguish T_0 and T_1 with non-negligible probability, then we can construct a PPT adversary $\mathcal{A}$ to break the multi-elements pseudorandomness of membership encryption scheme. $\mathcal{A}$ works as follows: when $\mathcal{A}$ receives ciphertexts $\{c_i^*\}_{i \in [n_\mathsf{x}]}$ from challenger, $\mathcal{A}$ executes as an honest receiver with the corrupt $\mathcal{S}$ except step 2. In this step, $\mathcal{A}$ computes OKVS as $D := \mathsf{Encode}_H(\{h(x_i), c_i^*\}_{i \in [n_\mathsf{x}]})$. Now $\mathcal{A}$ invokes $\mathcal{D}$ with the sender's view in the above interaction and outputs $\mathcal{D}$'s output. Note that if $\{c_i^*\}_{i \in [n_\mathsf{x}]}$ are the encryption of x_i, the view of corrupt sender is exactly the real view, which corresponds to T_0. If $\{c_i^*\}_{i \in [n_\mathsf{x}]}$ are random ciphertexts, the view corresponds to T_1. Therefore, $\mathcal{A}$ can break the multi-elements pseudorandomness of the membership encryption scheme with the same advantages as $\mathcal{D}$.
- Hybrid$_2$. Let T_2 be the same as T_1, except that the inputs of the receiver $\mathcal{R}$ are replaced by n_x random items. Note that the selection of value in OKVS has been replaced with random ciphertexts in T_1. By the obliviousness property of OKVS, T_1 and T_2 are statistically indistinguishable.
- Hybrid$_3$. Let T_3 be the same as T_2, except that the VOD execution is replaced by simulator $\mathsf{Sim}_{\mathsf{vod}}^S$. The security of VOD functionality guarantees the view is indistinguishable from real execution.

Corrupt receiver: $\mathsf{Sim}_{\mathcal{R}}(X = \{x_1, \ldots, x_{n_\mathsf{x}}\}, b)$ simulates the view of corrupt semi-honest receiver. It executes as follows:

1. $\mathsf{Sim}_{\mathcal{R}}$ generates a random encryption key k as the semi-honest receiver does in the real protocol. Then, it invokes VOD simulator $\mathsf{Sim}_{\mathsf{vod}}^R(k, b)$ and appends the output to the view.

The view output by $\mathsf{Sim}_{\mathcal{R}}$ is indistinguishable from the real one by the underlying simulators' indistinguishability.

Parameters:

- Two parties: sender $\mathcal{S}$ and receiver $\mathcal{R}$.
- Ideal $\mathcal{F}_{\mathsf{vod}}$ primitives specified in Fig. 14.
- A ME scheme $(\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$.
- An OKVS scheme $(\mathsf{Encode}_H, \mathsf{Decode}_H)$
- A collision-resistant hash function $h(x) : \{0,1\}^* \to \{0,1\}^\sigma$.

Input of $\mathcal{S}$: $Y = \{y_1, \ldots, y_{n_\mathsf{y}}\} \subset \{0,1\}^*$
Input of $\mathcal{R}$: $X = \{x_1, \ldots, x_{n_\mathsf{x}}\} \subset \{0,1\}^*$
Protocol:

1. $\mathcal{R}$ uses a ME scheme to generate a random key: $pp \leftarrow \mathsf{Setup}(X, 1^\kappa), k \leftarrow \mathsf{KeyGen}(pp)$ and computes $s_i := \mathsf{Enc}(k, x_i)$ for $i \in [n_\mathsf{x}]$.
2. $\mathcal{R}$ computes an OKVS $D := \mathsf{Encode}_H((h(x_1), s_1), \ldots, (h(x_{n_\mathsf{x}}), s_{n_\mathsf{x}}))$.
3. $\mathcal{R}$ sends D to the sender $\mathcal{S}$.
4. $\mathcal{S}$ computes $s_i^* := \mathsf{Decode}_H(D, h(y_i))$ for $i \in [n_\mathsf{y}]$.
5. $\mathcal{S}$ and $\mathcal{R}$ invoke the vector oblivious decryption functionality $\mathcal{F}_{\mathsf{vod}}$. $\mathcal{S}$ acts as sender with input $S = \{s_1^*, \ldots, s_{n_\mathsf{y}}^*\}$ and $\mathcal{R}$ acts as receiver with input k. As a result, $\mathcal{S}$ receives nothing and $\mathcal{R}$ receives $b \in \{0,1\}^{n_\mathsf{y}}$.

Fig. 15: ME-based Multi-Query Reverse Private Membership Test Protocol $\Pi_{\mathsf{mq\text{-}rpmt}}$

G Implementation Detail

G.1 Detailed Protocol Buffer Definition

The detailed protocol buffer definition is as follows:

```protobuf
syntax = "proto3";
message DataPacketProto {
    // the package contains head and payload, separately
    //defined by DataPacketSpecProto and PacketProto.
    HeaderProto headerProto = 1;
    PayloadProto payloadProto = 2;
    // head definition
    message HeaderProto {
        // task ID
        int64 taskId = 1;
        // protocol ID
        int32 ptoId = 2;
        // step ID
        int32 stepId = 3;
        // extra information
        int64 extraInfo = 4;
        // sender ID
        int32 senderId = 5;
        // receiver ID
        int32 receiverId = 6;
    }
    // payload definition
    message PacketProto {
        // repeated means the payload contains
        // an array of byte[]
        repeated bytes payloadBytes = 1;
    }
}
```

```
'''
```

G.2 JNI Technique

As mentioned in Section 6.2, we use the Java Native Interface (JNI) technique to invoke C/C++ libraries for speeding up performances. These include:

- Bit matrix transpose (used in OT extension and SKE-PSU). We follow the ideas provided by Mischasan[11] and adjust the implementation given in EMP-toolkit[12] to implement bit matrix transpose operations. The bit matrix is represented in the big-endian byte ordering, thus compatible with Java.
- Polynomial operations (used in KRTW and GMRSS). We tried the pure-Java Rings polynomial implementation[13] but found that its efficiency is not acceptable. We instead use the NTL library[14] with GMP library and GF2X library[15] for speeding up the performance. We adjust the polynomial representation to make the results returned from NTL compatible with Rings.
- ECC operations (used in base OT and PKE-PSU). We compared the ECC operation performances via different libraries, including the pure-Java Bouncy Castle[16], the C/C++ Relic[17], and the C/C++ MCL[18]. We found that (at least in our experiment platform) MCL library performs best, especially for the fixed-point multiplication operation. However, the ECC addition operation in Bouncy Castle is faster than MCL in our platforms. Therefore, we adjust the ECC point representation returned from MCL to make it compatible with the ECC point representation in Bouncy Castle to directly use Bouncy Castle to do the addition operations in Java.
- Switching Network programming (used in GMRSS). We used the code base open-sourced by Garimella et al. [GMR+21][19] as a starting point. We replaced the switching node representation from 'int' to 'int8_t' to reduce the memory cost.

[11] https://mischasan.wordpress.com/2011/10/03/the-full-sse2-bit-matrix-transpose-routine/
[12] https://github.com/emp-toolkit/emp-tool/blob/master/emp-tool/utils/block.h
[13] (https://rings.readthedocs.io/
[14] https://libntl.org/
[15] https://gitlab.inria.fr/gf2x/gf2x
[16] (https://www.bouncycastle.org/java.html
[17] https://github.com/relic-toolkit/relic
[18] https://github.com/herumi/mcl
[19] https://github.com/osu-crypto/PSI-analytics/blob/master/psi_analytics_eurocrypt19/common/benes.cpp