

UNITED: Shuffle-Free Multiparty PSU via Oblivious Invertible Sketches

Truong Son Nguyen
Arizona State University

Lucas Piske
Arizona State University

Ni Trieu
Arizona State University

Abstract

Multiparty Private Set Union (MPSU) allows multiple parties to compute the union of their private sets while revealing no additional information beyond the prescribed output. Existing practical MPSU protocols typically manipulate individual elements and rely on a distributed shuffle to hide element ownership, introducing an $O(n)$ round-complexity bottleneck.

We present UNITED, a shuffle-free MPSU framework based on a new data-structure abstraction, *Oblivious Insert-Only Invertible Sketches* (OIIS). Rather than anonymizing an element-wise intermediate representation, the parties directly construct a protected symmetric sketch of the union. Conditional oblivious insertion ensures that each distinct union element is inserted exactly once while hiding the insertion condition and modified locations.

We provide two realizations of OIIS. Our HE-based construction instantiates OIIS with a standard Invertible Bloom Lookup Table protected by multi-key partially homomorphic encryption, while our MPC-based construction uses additive secret sharing and introduces a new row-local power-sum IIS technique for efficient conditional insertion. We implement and evaluate both constructions. Across the evaluated configurations, our MPC-based construction achieves up to $3.10\times$ lower runtime in LAN settings, while our HE-based construction achieves up to $2.73\times$ lower runtime in WAN settings compared with prior MPSU protocols.

1 Introduction

Multiparty Private Set Union (MPSU) allows multiple parties, each holding a private set, to jointly compute the union of their sets without revealing information beyond the prescribed output. In the standard setting, the union is revealed only to a designated receiver. MPSU is useful in applications such as IP blacklist aggregation, federated submodule learning, private identity functionality, and joint graph computation.

Recent practical MPSU protocols [4, 6, 7, 12] typically follow a common three-stage structure. First, the parties perform

pairwise secret-shared private membership tests (SSPMTs) to determine which elements have already appeared in another party’s set. Second, the parties use the resulting secret shares to remove duplicate contributions through a message-modification step. Finally, they execute a distributed shuffle before reconstructing the output, thereby hiding the origin of each surviving element as shown in Figure 1. Although this approach achieves good overall performance, the distributed shuffle remains a major bottleneck. In particular, existing shuffle constructions process the parties sequentially, resulting in $O(n)$ communication rounds, where n denotes the number of parties, and making the protocol highly sensitive to network latency. As shown in Figure 2b, shuffle and decryption account for 44.6–80.7% of the total runtime of current state-of-the-art MPSU protocols across the evaluated network settings.

The need for shuffling stems from the fact that these protocols maintain an element-wise intermediate representation. Once individual elements are reconstructed, their positions or associated shares may reveal which party contributed them unless the elements are first anonymized. This raises a natural question: rather than anonymizing individual elements after processing them, can the parties construct an encoded representation in which ownership information is hidden by construction?

This motivates the central question of our work: “*Can we design an efficient state-of-the-art MPSU protocol without a distributed shuffle?*”

Technical Overview. We answer this question by introducing UNITED, a shuffle-free MPSU framework based on a new abstraction, *Oblivious Insert-Only Invertible Sketches* (OIIS). The key idea is to construct protected local sketches that can later be aggregated into a symmetric representation of the union. To determine which elements should be inserted, we introduce a simple building block called the *prefix membership test*, which determines, in secret-shared form, whether an element has already appeared in the union of the preceding parties’ sets. For each party P_j , the parties use this primitive to test whether each element of X_j belongs to the prefix union

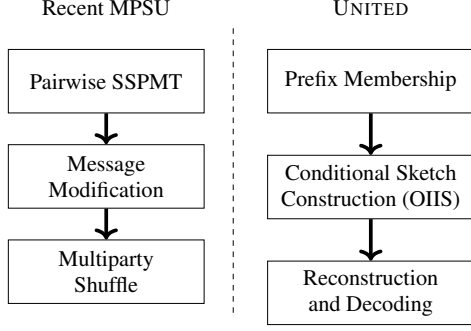


Figure 1: Comparison of the protocol flow of recent single-receiver element-wise MPSU protocols [4, 6, 7, 12] and UNITED. Recent protocols use message modification followed by a multiparty shuffle, whereas UNITED constructs a protected aggregate sketch through conditional insertion and then reconstructs and decodes the sketch.

$\bigcup_{i < j} X_i$. Party P_j then conditionally inserts an element into its local protected sketch only when it is the first party containing that element. Consequently, across all local sketches, every distinct union element is encoded exactly once.

The protected local sketches are then aggregated without revealing their contents. Because the underlying sketch representation is symmetric, the resulting aggregate sketch is independent of which party contributed each element and therefore hides element ownership without requiring a subsequent shuffle. The aggregate sketch is finally reconstructed and decoded to recover the union.

The use of an IIS also enables a natural optimization when the union cardinality is known. Since the size of the underlying invertible sketch depends on the number of distinct elements that must be encoded, UNITED can size the aggregate sketch according to the actual encoded set size rather than the worst-case bound $m \times n$, where m denotes the set size of each party and n the number of parties. This reduces computation and communication when the parties’ sets overlap substantially. In contrast, recent element-wise MPSU protocols process a fixed number of candidate elements determined by the input sizes and therefore cannot directly benefit from a smaller union in the same way.

Multi-Receiver MPSU. While revealing the union only to a designated receiver minimizes the information disclosed by the functionality, some applications naturally require all participating parties to obtain the final union. Examples include collaborative threat-intelligence and IP-blacklist aggregation, as well as consortium-wide vulnerability sharing [4, 7]. This functionality also appears in the two-party setting; for example, the IBLT-based PSU protocol of [13] reveals the union to both parties. Motivated by these settings, we also consider a multi-receiver variant of UNITED in which the union is intentionally revealed to all parties. The sketch-based representation makes this extension natural: the aggregate sketch

can be partitioned across the parties, allowing reconstruction and decoding to be distributed rather than performed entirely by a single receiver.

Figure 1 compares the workflow of recent single-receiver, element-wise MPSU protocols [4, 6, 7, 12] with UNITED. By eliminating the sequential $O(n)$ -round shuffle, UNITED removes the main round-complexity bottleneck of prior protocols. Its overall communication-round complexity is $O(\log n)$, dominated by the prefix-membership stage, as summarized in Table 2a. A detailed discussion of related work is provided in Appendix A.

Design Challenges. Realizing this idea efficiently requires addressing two main challenges.

- **Conditional insertion.** The private membership-test stage determines whether an element has already appeared, but the parties must use this result to update the encoded representation without revealing the membership bit or whether the insertion takes effect.
- **Private addressing.** Updating an encoded representation may reveal the locations associated with the inserted element. How this problem is addressed depends on the cryptographic representation of the sketch.

We address both challenges through the OIS abstraction. OIS allows an element to be conditionally inserted into a protected invertible sketch while hiding the insertion condition and the information associated with the update. While generic oblivious data structures [10, 19] could provide similar access-pattern hiding, they support substantially more general operations than required here; OIS instead exploits the restricted insert-only structure of invertible sketches to obtain more efficient realizations.

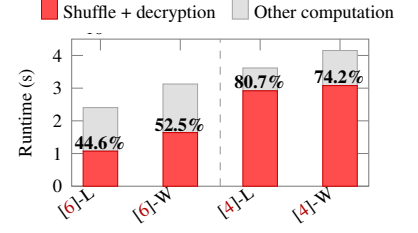
We provide two OIS realizations with different performance characteristics. In the HE-based realization, the data owner holds an encrypted sketch and can locally determine the locations associated with its own elements, while the insertion condition remains encrypted throughout the homomorphic evaluation. In the MPC-based realization, the sketch is additively secret-shared, so private addressing must also be handled inside the secure computation. To make this efficient, we introduce a row-local power-sum IIS that restricts each insertion to a small row-local representation rather than requiring an update vector over the entire sketch.

Contributions. Our contributions are summarized as follows.

- **A shuffle-free framework for MPSU.** We identify secure conditional insertion into a symmetric encoded representation as the key operation needed to eliminate the distributed shuffle. Based on this observation, we introduce the OIS abstraction and use it to build UNITED, a generic MPSU framework in which each distinct union element is encoded exactly once. The resulting aggregate

Protocol	Pairwise SSPMT	Prefix Membership	Message Modification	OIIS	Reconst. & Decrypt.	Shuffle	Total
GNT24 [7]	$O(n)$	–	$O(n)$	–	$O(n)$	$O(n)$	$O(n)$
DCZB25 [4]	$O(1)$	–	$O(n)$	–	$O(n)$	$O(n)$	$O(n)$
PULSE [6]	$O(1)$	–	$O(1)$	–	$O(1)$	$O(n)$	$O(n)$
UNITED	–	$O(\log n)$	–	$O(1)$	$O(1)$	–	$O(\log n)$

(a) Communication-round complexity.



(b) Runtime breakdown.

Figure 2: Comparison with prior MPSU protocols. (a) Communication-round complexity. (b) Runtime breakdown showing the fraction spent on shuffle and decryption for PULSE [6] and DCZB [4] for $m = 2^{16}$ and $n = 8$, where L and W denote LAN and WAN 1, respectively.

sketch hides element ownership by construction and removes the need for message modification and multiparty shuffling. UNITED supports both the standard single-receiver setting and a multi-receiver setting in which the union is revealed to all parties; for the latter, we develop distributed reconstruction techniques that partition the final sketch workload across the parties.

- **Two efficient OIIS realizations.** We instantiate OIIS using two cryptographic approaches. Our HE-based construction combines a standard IBLT with multi-key partially homomorphic encryption and requires only lightweight homomorphic additions and ciphertext-plaintext multiplications. Our MPC-based construction uses additive secret sharing together with a new row-local power-sum IIS, reducing the private addressing domain from the entire sketch to a small row-local representation. We believe the OIIS abstraction and its realizations may be of independent interest beyond MPSU.
- **Union-size-aware optimization.** When the union cardinality is known, UNITED can reduce the size of its invertible sketch according to the actual number of distinct elements rather than the worst-case bound $m \times n$. For the MPC-based realization, we additionally exploit exact row occupancies in the multi-receiver setting. This reduces computation and communication when the parties' sets overlap substantially, yielding up to a $2\times$ runtime reduction for the MPC-based realization and up to a $2.36\times$ reduction for the HE-based realization.
- **Performance evaluation.** We implement both realizations and evaluate their runtime and communication cost across different set sizes, numbers of parties, and LAN and WAN network conditions. Compared with prior state-of-the-art MPSU protocols, our MPC-based construction achieves up to a $3.10\times$ runtime improvement in LAN settings, while our HE-based construction achieves up to a $2.73\times$ improvement in WAN settings. When the union cardinality is known, UNITED can achieve additional performance improvements through union-size-aware parameter selection.

2 Preliminaries

In this work, the computational and statistical security parameters are denoted by κ and λ , respectively. We use $[\cdot]$ notation to denote index sets. For example, $[m]$ denotes the set $\{0, \dots, m-1\}$, while $[i, j]$ denotes the set $\{i, \dots, j-1\}$. For a sketch state T and a set of row indices $\mathcal{R}$, we write $T|_{\mathcal{R}}$ to denote the restriction of T to the rows indexed by $\mathcal{R}$.

For additive secret sharing over a field $\mathbb{F}$, we write $\langle x \rangle = (\langle x \rangle_1, \dots, \langle x \rangle_n)$ to denote a sharing of $x \in \mathbb{F}$ among parties $P_1, \dots, P_n$, where party P_i holds $\langle x \rangle_i$ and $x = \sum_{i=1}^n \langle x \rangle_i$. Addition of secret-shared values is performed locally and requires no communication: given $\langle x \rangle$ and $\langle y \rangle$, each party computes $\langle x + y \rangle_i = \langle x \rangle_i + \langle y \rangle_i$. Multiplication of a secret-shared value by a public scalar is also local. Multiplication of two secret-shared values is interactive and is realized using Beaver multiplication triples [1].

In this paper, we consider the semi-honest threat model with a dishonest majority. Parties follow the prescribed protocol but may try to learn additional information from their views. We allow any coalition of up to $n-1$ parties to collude and share their inputs, randomness, and protocol views. Such a coalition should learn no information about the honest parties' inputs beyond what is implied by the prescribed output. This threat model is commonly considered in prior MPSU works such as [4, 6].

MPSU and Union Cardinality The multiparty private set union (MPSU) functionality allows n parties $P_1, \dots, P_n$, holding private sets $X_1, \dots, X_n$, to compute their set union. We consider two variants: a single-receiver functionality, where only party P_1 receives the union, and a multi-receiver functionality, where every party receives the union. Both variants are specified in Figure 3. The MPSU cardinality functionality, shown in Figure 4, outputs only the union cardinality $C = |\bigcup_{i=1}^n X_i|$ to all parties.

Multi-Key Cryptosystem. For the public-key-based realization, we use a multi-key cryptosystem following [6]. Key generation outputs a joint public key pk and secret keys

Parameters: n parties $P_1, \dots, P_n$.

Behavior:

- Wait to receive a set X_i from every party $P_i, i \in [n]$.
- Compute the union $U = \bigcup_{i=1}^n X_i$.
- Output U to party P_1 .
- **Multi-receiver variant:** Output U to all $P_{i \in [n]}$.

Figure 3: Ideal MPSU ($\mathcal{F}_{\text{MPSU}}$) functionality. The highlighted line specifies the multi-receiver variant.

Parameters: n parties $P_1, \dots, P_n$.

Behavior:

- Wait to receive a set X_i from every party $P_i, i \in [n]$.
- Compute the union cardinality $C = |\bigcup_{i=1}^n X_i|$.
- Output C to every party $P_i, i \in [n]$.

Figure 4: Ideal MPSU cardinality ($\mathcal{F}_{\text{MPSU-CA}}$) functionality.

$sk_1, \dots, sk_n$. Ciphertexts under pk support partial decryption by individual parties and full decryption using the required secret keys. We additionally require partial homomorphic evaluation supporting ciphertext-ciphertext addition, plaintext-ciphertext addition, and plaintext-ciphertext multiplication. We use “evaluate homomorphically” to refer to these operations. The full definition is given in Appendix C.1; a concrete EC-ElGamal realization appears in [6].

Secret-Sharing Private Membership Test A secret-sharing private membership test involves a set holder S with set X and a query holder Q with element x , who receive bit shares b_S, b_Q such that $b_S \oplus b_Q = \mathbf{1}[x \in X]$. Following [4], we use the batched variant in which S holds B disjoint sets $\{X_{S,1}, \dots, X_{S,B}\}$ and Q holds corresponding queries $\{x_{Q,1}, \dots, x_{Q,B}\}$. We denote this functionality by $\mathcal{F}_{\text{bSSPMT}}$; its ideal functionality is shown in Figure 5.

Additive Zero-Sharing The additive zero-sharing functionality allows n parties to obtain fresh additive shares of zero. The shares are individually random, while their sum equals zero. The functionality can be realized using a simple and efficient protocol based on [3], costing each party $O(n)$ field communication offline to send a seed to others, and $O(mn)$ during online phase to extend the seed using PRG. Its ideal functionality is shown in Figure 6.

Invertible Bloom Lookup Tables (IBLTs) IBLTs [9] are probabilistic data structures that support recovery of inserted elements with high probability. An IBLT uses multiple bins addressed by a small number of hash functions and supports linear insertion and deletion. Decoding iteratively peels bins

Parameters: Set holder S , query holder Q , batch size B , message length ℓ , and set size m .

Behavior:

- Wait to receive B disjoint sets $\{X_{S,1}, \dots, X_{S,B}\}$ from S , where $X_{S,i} \subseteq ([0,1]^\ell)^m$ for every $i \in [B]$.
- Wait to receive B queries $\{x_{Q,1}, \dots, x_{Q,B}\}$ from Q , where $x_{Q,i} \in [0,1]^\ell$ for every $i \in [B]$.
- For every $i \in [B]$, sample and output bit shares $b_{S,i}$ to S and $b_{Q,i}$ to Q such that $b_{S,i} \oplus b_{Q,i} = 1$ if $x_{Q,i} \in X_{S,i}$ and $b_{S,i} \oplus b_{Q,i} = 0$ otherwise.

Figure 5: Batched secret-shared private membership test ($\mathcal{F}_{\text{bSSPMT}}$) functionality.

Parameters: n parties $P_1, \dots, P_n$ and an additive sharing domain $\mathbb{F}$.

Behavior:

- Sample additive shares $\langle 0 \rangle_1, \dots, \langle 0 \rangle_n \in \mathbb{F}$ such that $\sum_{i=1}^n \langle 0 \rangle_i = 0$.
- Output $\langle 0 \rangle_i$ to party P_i , for every $i \in [n]$.

Figure 6: Additive zero-sharing ($\mathcal{F}_{\text{ZeroShare}}$) functionality.

containing a single recoverable element. With appropriate sizing, all encoded elements are recovered with high probability. We use the parameters of [13]; details are given in Appendix C.2.

Secret-Sharing Private Prefix Membership Test. We define a private prefix membership test (PPMT) for a query party $P_Q, Q > 1$, holding x_Q , which tests whether $x_Q \in \bigcup_{i=1}^{Q-1} X_i$. In the secret-shared variant (SSPPMT), each party $P_i, i \in [1, Q]$, receives an additive share b_i such that $\sum_{i=1}^Q b_i = \mathbf{1}[x_Q \in \bigcup_{i=1}^{Q-1} X_i]$. In this work, we use a batched multi-query variant (mq-SSPPMT), where P_Q provides queries $x_{Q,1}, \dots, x_{Q,B}$ and the parties receive additive shares $b_{i,v}$ satisfying $\sum_{i=1}^Q b_{i,v} = \mathbf{1}[x_{Q,v} \in \bigcup_{i=1}^{Q-1} X_i]$ for every $v \in [B]$. Its ideal functionality $\mathcal{F}_{\text{mq-SSPPMT}}$ is shown in Figure 7.

3 Insert-Only Invertible Sketches

Our MPSU framework does not fundamentally depend on a particular data structure such as an IBLT. Instead, it requires an encoded representation that supports incremental insertion of elements and subsequent recovery of the inserted set. We capture this requirement through a general data-structure abstraction that we call an *Insert-Only Invertible Sketch* (IIS).

Parameters: Set holders $P_1, \dots, P_{Q-1}$, query party P_Q , and batch size B .

Behavior:

- $\forall i \in [1, Q-1]$, wait to receive a set X_i from party P_i .
- Wait to receive a batch of queries $\{x_{Q,1}, \dots, x_{Q,B}\}$ from party P_Q .
- $\forall v \in [B]$, sample additive shares $b_{1,v}, \dots, b_{Q,v}$ and output $b_{i,v}$ to party P_i , for every $i \in [1, Q]$, such that

$$\sum_{i=1}^Q b_{i,v} = \mathbf{1} \left[x_{Q,v} \in \bigcup_{i=1}^{Q-1} X_i \right].$$

Figure 7: Multi-query secret-shared private prefix membership test ($\mathcal{F}_{\text{mq-SSPPMT}}$) functionality.

3.1 Definition

Definition 1 (Insert-Only Invertible Sketch). *An Insert-Only Invertible Sketch (IIS) is a tuple of algorithms $\text{IIS} = (\text{Insert}, \text{Decode})$ operating on a sketch state T , where*

- $T' \leftarrow \text{IIS.Insert}(x, T)$ inserts an element x into the sketch;
- $X \leftarrow \text{IIS.Decode}(T)$ recovers the elements encoded in T .

We additionally require insertion to be symmetric: for every sketch state T and every pair of elements x, y ,

$$\text{IIS.Insert}(x, \text{IIS.Insert}(y, T)) = \text{IIS.Insert}(y, \text{IIS.Insert}(x, T)).$$

The symmetry property ensures that the resulting sketch is independent of the insertion order. Consequently, insertion naturally extends from individual elements to sets. For $X = \{x_1, \dots, x_m\}$, we define $\text{IIS.Insert}(X, T) = \text{IIS.Insert}(x_1, \text{IIS.Insert}(x_2, \dots \text{IIS.Insert}(x_m, T)))$.

This property is useful when multiple parties, streams, or distributed sources contribute elements to a common encoded representation. For MPSU, it allows the parties to construct a symmetric representation of the union directly. Unlike element-wise constructions of previous work [4, 6] that require a subsequent shuffle to hide element ownership, the resulting sketch is independent of the order in which parties contribute their elements.

An IBLT is one concrete realization of an IIS. Let T consist of M cells and let $h_1, \dots, h_k$ be public hash functions. In the simplified representation used in this work, each cell stores a key field and a cnt field. To insert x , the IBLT updates each cell $h_j(x)$, $j \in [k]$, as $T[h_j(x)].\text{key} \leftarrow T[h_j(x)].\text{key} + x$ and $T[h_j(x)].\text{cnt} \leftarrow T[h_j(x)].\text{cnt} + 1$. The corresponding decoder repeatedly identifies cells with $\text{cnt} = 1$, recovers the remaining element from the key field, and peels its contribution from the associated cells. Hence, standard IBLT insertion realizes IIS.Insert and IBLT peeling realizes IIS.Decode .

Oblivious IIS. The ordinary IIS abstraction does not provide privacy: directly executing $\text{IIS.Insert}(x, T)$ may reveal the inserted element, the modified locations, or both. We therefore define an *Oblivious Insert-Only Invertible Sketch* (OIIS) as a cryptographically protected realization of IIS in which insertions are performed without revealing the private information associated with the update.

Generic oblivious data structures [10, 19] could in principle be used to hide the access pattern of IIS updates. However, these constructions are designed to support general oblivious reads and writes and therefore rely on generic oblivious-memory machinery with polylogarithmic overhead. In contrast, OIIS only requires conditional insertions into an algebraically structured sketch. By exploiting this restricted update pattern, our realizations avoid generic oblivious-access mechanisms and operate directly on the sketch representation.

In our MPSU application, we specifically require *conditional oblivious insertion*, since each distinct element in the union should be inserted exactly once. For an element $x \in X_j$, define $b_x = \mathbf{1}[x \in \bigcup_{i < j} X_i]$. The insertion is performed only when $b_x = 0$, i.e., when P_j is the first party containing x .

Definition 2 (Conditional OIIS). *Let $\mathcal{T}$ be an IIS state represented in a secure form, let x be an element held by a designated data owner, and let $b \in \{0, 1\}$ be a private condition bit. Conditional OIIS insertion outputs a protected state $\mathcal{T}^*$ such that*

$$\mathcal{T}^* \equiv \begin{cases} \text{IIS.Insert}(x, \mathcal{T}), & b = 0, \\ \mathcal{T}, & b = 1, \end{cases}$$

where $\equiv$ denotes equality of the underlying plaintext IIS states.

We say that the insertion is oblivious if the protected representation reveals no information beyond the prescribed leakage. In particular, observing the secure representation of the sketch before and after the operation should not reveal the inserted element, the condition bit, or whether and where the underlying sketch was updated.

The secure representation depends on the underlying cryptographic realization. In the following subsection, we instantiate OIIS using encrypted and secret-shared representations.

More generally, OIIS may support applications in which repeated insertions are meaningful. This work, however, focuses on MPSU, where the output should reveal only the set union. If the same element is inserted multiple times, its multiplicity in the reconstructed sketch may reveal how many parties hold that element, leaking information about the parties' intersections beyond the union itself. We therefore require each distinct element to be inserted exactly once through conditional oblivious insertion. Henceforth, unless stated otherwise, we use OIIS to refer to this conditional OIIS functionality.

Figure 8 defines the batched conditional OIIS functionality used throughout our MPSU protocols. Party P_Q provides a set

Parameters: Q parties $P_1, \dots, P_Q$ and set size m .

Behavior:

- Wait to receive a set $X_Q = \{x_1, \dots, x_m\}$ from P_Q .
- For every $x_q \in X_Q$, wait to receive from each party P_i , $i \in [Q]$, an additive share $\langle b_q \rangle_i$ such that $b_q = \sum_{i=1}^Q \langle b_q \rangle_i \in \{0, 1\}$, where $b_q = 0$ indicates insertion and $b_q = 1$ indicates that the insertion is skipped.
- Initialize an empty IIS state $T_Q \leftarrow \mathbf{0}$.
- Compute $T_Q \equiv \text{IIS.Insert}(\{x_q \in X_Q \mid b_q = 0\}, T_Q)$.
- Output the protected representation $\tilde{T}_Q$ of T_Q to the parties designated by the underlying realization.^a

^aIn the HE-based realization, $\tilde{T}_Q = \text{Enc}_{\text{pk}}(T_Q)$ is output to P_Q . In the MPC-based realization, each party P_i , $i \in [Q]$, receives a share $\langle T_Q \rangle_i$ such that $T_Q = \sum_{i=1}^Q \langle T_Q \rangle_i$.

Figure 8: Conditional OIIS Encoding ($\mathcal{F}_{\text{OIIS}}$) Functionality.

X_Q , while the participating parties hold additive shares of one conditional-insertion bit for each element. The functionality inserts exactly those elements whose condition bit evaluates to zero and returns a protected representation $\tilde{T}_Q$ of the resulting IIS state.

The concrete representation of $\tilde{T}_Q$ depends on the underlying cryptographic realization. In the HE-based construction, $\tilde{T}_Q = \text{Enc}_{\text{pk}}(T_Q)$, where pk is the joint public key. In the MPC-based construction, $\tilde{T}_Q = \langle T_Q \rangle = (\langle T_Q \rangle_1, \dots, \langle T_Q \rangle_Q)$, where $T_Q = \sum_{i=1}^Q \langle T_Q \rangle_i$. Thus, both constructions realize the same batched conditional OIIS functionality while differing only in how the resulting sketch state is cryptographically protected.

Once the underlying IIS state is reconstructed in the clear, decoding proceeds according to the corresponding IIS decoding algorithm.

3.2 Protocols

We present two realizations of OIIS: an HE-based construction using a standard IBLT, and a secret-sharing-based MPC construction using a new data structure that we call the *Row-Local Power-Sum IIS*.

3.2.1 HE-Based OIIS

We first instantiate OIIS using a standard IBLT as the underlying IIS and protect its state with a multi-key partially homomorphic encryption scheme. The scheme provides a joint public key pk shared by all parties, while each party P_i holds a corresponding secret-key share sk_i used for collaborative decryption. Party P_Q maintains the entire encrypted sketch and can locally determine the cells associated with each element $x \in X_Q$.

Parameters: Q parties $P_1, \dots, P_Q$, a multi-key partially homomorphic encryption scheme with joint public key pk , and a standard IIS instantiated with an IBLT using hash functions $h_1, \dots, h_k$.

Input:

- Party P_Q holds a set $X_Q = \{x_1, \dots, x_m\}$.
- For every $x_q \in X_Q$, party P_i , $i \in [Q]$, holds an additive share $\langle b_q \rangle_i$ such that $b_q = \sum_{i=1}^Q \langle b_q \rangle_i \in \{0, 1\}$.

Protocol:

1. Party P_Q initializes an empty encrypted IIS state $\tilde{T}_Q \leftarrow \text{Enc}_{\text{pk}}(\mathbf{0})$.
2. For every $x_q \in X_Q$, each party P_i , $i \in [Q]$, encrypts $\langle b_q \rangle_i$ under pk and sends $\text{Enc}_{\text{pk}}(\langle b_q \rangle_i)$ to P_Q .
3. For every $x_q \in X_Q$, party P_Q homomorphically computes $\text{Enc}_{\text{pk}}(b_q) = \sum_{i=1}^Q \text{Enc}_{\text{pk}}(\langle b_q \rangle_i)$.
4. Party P_Q locally computes the affected IIS locations $h_1(x_q), \dots, h_k(x_q)$.
5. For every $r \in \{h_1(x_q), \dots, h_k(x_q)\}$, party P_Q updates

$$\text{cnt}_r \leftarrow \text{cnt}_r + 1 - \text{Enc}_{\text{pk}}(b_q)$$

$$\text{key}_r \leftarrow \text{key}_r + x_q - x_q \cdot \text{Enc}_{\text{pk}}(b_q).$$
6. After processing all elements in X_Q , party P_Q rerandomizes all ciphertexts in $\tilde{T}_Q$.

Output: Party P_Q outputs $\tilde{T}_Q = \text{Enc}_{\text{pk}}(T_Q)$, a freshly randomized encryption of the resulting IIS state.

Figure 9: HE-based batched OIIS encoding $\Pi_{\text{HE-OIIS}}$.

Conditional Oblivious Encoding. For each $x \in X_Q$, every party P_i , $i \in [Q]$, encrypts its share $\langle b \rangle_i$ under the joint public key pk and sends $\text{Enc}_{\text{pk}}(\langle b \rangle_i)$ to P_Q . Using homomorphic addition, P_Q computes $\text{Enc}_{\text{pk}}(b) = \sum_{i=1}^Q \text{Enc}_{\text{pk}}(\langle b \rangle_i) = \text{Enc}_{\text{pk}}(\sum_{i=1}^Q \langle b \rangle_i)$.

Since P_Q owns x , it can locally determine the affected IBLT locations $h_1(x), \dots, h_k(x)$. For each corresponding row, P_Q performs $\text{cnt} \leftarrow \text{cnt} + 1 - \text{Enc}_{\text{pk}}(b)$ and $\text{key} \leftarrow \text{key} + x - x \cdot \text{Enc}_{\text{pk}}(b)$. When $b = 0$, these updates are exactly a standard IIS insertion; when $b = 1$, the added contribution evaluates to zero and the underlying plaintext table remains unchanged.

Because the update locations are computed locally by the data owner, the HE-based realization does not require the oblivious-addressing mechanism needed in the MPC-based construction discussed later. The condition bit remains encrypted throughout the homomorphic evaluation. After processing all elements in X_Q , party P_Q rerandomizes all ciphertexts in the table, producing a fresh encryption of the same underlying IIS state. The formal HE-based encoding protocol is presented in Figure 9.

Correctness and Security. The correctness of $\Pi_{\text{HE-OIS}}$ follows directly from the homomorphic update. For each element x_q , the protocol performs a standard IBLT insertion when $b_q = 0$, while the added contribution evaluates to zero when $b_q = 1$.

Security relies on the privacy of the additive shares and the security of the underlying multi-key partially homomorphic encryption scheme. Since P_Q already owns x_q , allowing it to compute the corresponding locations $h_1(x_q), \dots, h_k(x_q)$ introduces no additional leakage. The condition bit remains encrypted throughout the evaluation, and rerandomization produces a fresh secure representation of the same underlying IIS state. Thus, the output ciphertexts cannot be linked to the intermediate ciphertext states observed during conditional insertion. We omit the formal proof of the following theorem.

Theorem 1. *Assuming the privacy of the additive secret-sharing scheme and the security and rerandomization properties of the underlying multi-key partially homomorphic encryption scheme, protocol $\Pi_{\text{HE-OIS}}$ correctly and securely realizes the batched conditional OIS encoding functionality against semi-honest adversaries with a dishonest majority.*

Decoding. Since the underlying IIS is instantiated with an IBLT, decoding is identical to standard IBLT decoding once the sketch is reconstructed in the clear by the designated party.

3.2.2 MPC-Based OIS

We next consider an MPC-based realization of OIS. Unlike the HE-based construction, where party P_Q holds the entire encrypted sketch and can locally determine the locations associated with an element x , the MPC realization stores the sketch in additive secret-shared form. Specifically, party P_i , $i \in [Q]$, holds a share $\langle T_Q \rangle_i$ such that $T_Q = \sum_{i=1}^Q \langle T_Q \rangle_i$. Hence, no individual party holds the complete sketch, and the parties must jointly perform each conditional insertion without revealing the private update locations.

A direct realization using a standard IBLT is inefficient in this setting. To insert an element x , party P_Q must specify the affected cells $h_1(x), \dots, h_k(x)$ without revealing these locations to the other parties. A potential solution is to encode the update locations as a length- M indicator vector and secret-share it among the participating parties. Since a standard IBLT has $M = O(m)$ cells, this incurs $O(m)$ communication and computation per element, including the corresponding MPC evaluation cost, and therefore $O(m^2)$ total cost over a set of size m .

Note that batching the updates does not resolve this issue. Although P_Q knows the hash locations of its own elements, the other parties only hold secret shares of the sketch and therefore cannot locally align their shares with these private locations. The parties would still require an oblivious mechanism for determining which global IBLT cells are updated.

Thus, while a standard IBLT works naturally in the HE-based realization, its global addressing pattern is poorly suited to a secret-sharing-based MPC realization.

To address this issue, we introduce a *Row-Local Power-Sum IIS* tailored to MPC. We choose $M = O\left(\frac{m}{\log m}\right)$ rows and a public hash function $h_0 : \{0, 1\}^\ell \rightarrow [M]$. Each element x is assigned to exactly one row $r = h_0(x)$. Under the standard balls-into-bins analysis [14], the maximum row occupancy is $L = O(\log m)$ with high probability, so the total number of row-slot positions is $ML = O(m)$. Thus, the updating domain is reduced from the entire sketch to a single row of size $L = O(\log m)$. This is crucial for the MPC realization: instead of representing each item insertion using a length- $M = O(m)$ global update vector, as would be required for the above IBLT-based approach, the data owner only needs to secret-share a row-local vector of length $O(L)$ per item. Consequently, the communication and MPC evaluation cost per item is reduced from linear in the sketch size to logarithmic in the set size.

To compactly represent all elements assigned to the same row, we store their power sums rather than the elements themselves. This allows a row containing up to L elements to be represented using only $L + 1$ field elements, while preserving sufficient information to recover all encoded elements during decoding. Moreover, the power-sum representation is additive: inserting a new element only requires adding its corresponding power vector to the current row state. This property makes the representation particularly suitable for secret-sharing-based MPC, where additions on secret shares can be performed locally without additional communication.

Concretely, for each row r , the IIS state is represented by $T_Q[r] = (S_{r,0}, S_{r,1}, \dots, S_{r,L})$, where $S_{r,0}$ records the number of encoded elements in the row and $S_{r,j} = \sum_{x \in T_Q[r]} x^j$, $j \in [L]$. For an element x assigned to row r , define the row-local power vector $e_r(x) = [1, x, x^2, \dots, x^L]$. Insertion is performed by vector addition, $T_Q[r] \leftarrow T_Q[r] + e_r(x)$. Because addition is commutative, the resulting sketch is independent of the order in which elements are inserted and therefore satisfies the symmetry requirement of IIS. Thus, each item insertion is represented by a row-local vector of length $L + 1$, rather than by a global update vector spanning the entire sketch.

Conditional Oblivious Encoding. To realize conditional insertion over this IIS in MPC, party P_Q organizes its set into a row-padded representation $\hat{X}_Q = \{\hat{x}_{r,s}\}_{r \in [M], s \in [L]}$, where each real element appears in row $h_0(x)$ and every unused position contains a padding element $\perp$.

For each position (r, s) , the parties hold additive shares $\langle b_{r,s} \rangle_i$ of a condition bit $b_{r,s} = \sum_{i=1}^Q \langle b_{r,s} \rangle_i \in \{0, 1\}$, where $b_{r,s} = 0$ indicates insertion and $b_{r,s} = 1$ skips the insertion. Note that the condition-bit shares can be aligned with the row-slot representation by the preceding private membership-test stage, which outputs the shares in the same row-slot ordering.

For each real element $\hat{x}_{r,s}$, party P_Q computes $e_r(\hat{x}_{r,s}) =$

Parameters: Q parties $P_1, \dots, P_Q$, M rows, row capacity L , and a public hash function $h_0 : \{0, 1\}^\ell \rightarrow [M]$.

Input:

- Party P_Q holds a set $X_Q = \{x_1, \dots, x_m\}$.
- For every $x_q \in X_Q$, party P_i , $i \in [Q]$, holds the corresponding additive share of the conditional-insertion bit.

Protocol:

1. The parties initialize an empty secret-shared IIS state $\langle T_Q \rangle \leftarrow \mathbf{0}$.
2. Party P_Q constructs a row-padded representation $\hat{X}_Q = \{\hat{x}_{r,s}\}_{r \in [M], s \in [L]}$, where every real element $x \in X_Q$ appears in row $h_0(x)$ and every unused position contains $\perp$. The corresponding condition-bit shares are arranged according to the same row-slot ordering.^a
3. For every $(r, s) \in [M] \times [L]$, party P_Q computes

$$e_r(\hat{x}_{r,s}) = \begin{cases} [1, \hat{x}_{r,s}, \hat{x}_{r,s}^2, \dots, \hat{x}_{r,s}^L], & \hat{x}_{r,s} \neq \perp, \\ \mathbf{0}, & \hat{x}_{r,s} = \perp, \end{cases}$$

and secret-shares $e_r(\hat{x}_{r,s})$ among the participating parties.

4. For every row $r \in [M]$ and every position $s \in [L]$, let $\langle b_{r,s} \rangle$ denote the additive sharing of the corresponding condition bit. The parties jointly compute an additive sharing $\langle \gamma_{r,s} \rangle$ of $\gamma_{r,s} = (1 - b_{r,s}) \cdot e_r(\hat{x}_{r,s})$ using arithmetic MPC.
5. For every row $r \in [M]$ and every position $s \in [L]$, each party locally updates its share of row r as $\langle T_Q[r] \rangle \leftarrow \langle T_Q[r] \rangle + \langle \gamma_{r,s} \rangle$.

Output: Each party P_i outputs its additive share $\langle T_Q \rangle_i$ of the resulting row-local power-sum IIS state.

^aThis alignment can be produced directly by the preceding SSPMT stage, which outputs the condition-bit shares in the same row-slot layout.

Figure 10: MPC-based batched OIIS encoding ($\Pi_{\text{MPC-OIIS}}$).

$[1, \hat{x}_{r,s}, \hat{x}_{r,s}^2, \dots, \hat{x}_{r,s}^L]$, while for a padding position it sets $e_r(\perp) = \mathbf{0}$. Party P_Q secret-shares $e_r(\hat{x}_{r,s})$ among the participating parties.

Since both $b_{r,s}$ and $e_r(\hat{x}_{r,s})$ are secret-shared, the parties jointly compute an additive sharing $\langle \gamma_{r,s} \rangle$ of $\gamma_{r,s} = (1 - b_{r,s}) \cdot e_r(\hat{x}_{r,s})$ using arithmetic MPC. They then update the shared row as $\langle T_Q[r] \rangle \leftarrow \langle T_Q[r] \rangle + \langle \gamma_{r,s} \rangle$. The multiplication requires MPC evaluation, whereas the final addition is local and incurs no additional communication under additive secret sharing.

Since every row-slot position is processed using the same MPC computation, no party other than P_Q learns which positions contain real elements, the corresponding condition bits, or which insertions take effect. The formal MPC-based encoding protocol is presented in Figure 10.

Correctness and Security. The correctness of $\Pi_{\text{MPC-OIIS}}$ follows directly from the conditional row update. For every row-slot position (r, s) , the parties compute $\gamma_{r,s} = (1 - b_{r,s}) \cdot e_r(\hat{x}_{r,s})$. Hence, when $b_{r,s} = 0$, the power vector $e_r(\hat{x}_{r,s})$ is added to the corresponding row, which is exactly a row-local IIS insertion; when $b_{r,s} = 1$, the contribution is zero and the sketch remains unchanged. Padding positions also contribute zero because $e_r(\perp) = \mathbf{0}$. Since row updates are additive and therefore commutative, the reconstructed sketch encodes exactly the elements whose condition bits evaluate to zero, independently of the insertion order. We show later that, given such a correctly encoded sketch, the decoding procedure recovers exactly the encoded set.

Security follows from the privacy of additive secret sharing and the security of the underlying arithmetic MPC protocol. The values $b_{r,s}$, $e_r(\hat{x}_{r,s})$, $\gamma_{r,s}$, and the sketch state T_Q remain secret-shared throughout the protocol. Since every row-slot position is processed using the same MPC computation, no party other than P_Q learns which positions contain real elements, the values of the corresponding condition bits, or which conditional insertions are performed. The outer row index itself need not be hidden: even if h_0 is public, all slots in every row are processed uniformly, so the observed access pattern reveals no additional information about whether a particular position contains a real element or whether its update takes effect. We provide a formal proof of the following theorem in Appendix D.2.

Theorem 2. *Assuming the privacy of the additive secret-sharing scheme and the security of the underlying arithmetic MPC protocol, $\Pi_{\text{MPC-OIIS}}$ correctly and securely realizes the batched conditional OIIS encoding functionality against semi-honest adversaries with a dishonest majority.*

Decoding. Once the IIS is reconstructed in the clear, decoding is performed independently for each row. Let $d_r = S_{r,0}$ denote the number of elements encoded in row r . The entries $S_{r,1}, \dots, S_{r,d_r}$ are the corresponding power sums.

The decoder applies Newton's identities to recover the elementary symmetric polynomials $e_{r,1}, \dots, e_{r,d_r}$, where $e_{r,0} = 1$ and $e_{r,k} = \frac{1}{k} \sum_{j=1}^k (-1)^{j-1} e_{r,k-j} S_{r,j}$. It then constructs the monic polynomial

$$f_r(Z) = Z^{d_r} - e_{r,1}Z^{d_r-1} + e_{r,2}Z^{d_r-2} - \dots + (-1)^{d_r}e_{r,d_r}.$$

By construction, $f_r(Z) = \prod_{x \in T_Q[r]} (Z - x)$, and therefore its roots are exactly the elements encoded in row r .

The decoder repeats this procedure for every row and takes the union of the recovered roots. Since each element is assigned to exactly one outer row by h_0 , successful decoding of every row yields exactly the set encoded in the global sketch. About the decoding cost, each row is decoded independently using Newton's identities and root finding on a polynomial of degree at most $L = O(\log m)$.

m	2^{12}	2^{14}	2^{16}	2^{18}	2^{20}	2^{22}	2^{24}
L	48	53	58	63	68	73	78
ML/m	4.01	3.79	3.63	3.50	3.40	3.32	3.25

Table 1: Concrete row-local IIS parameters for statistical error 2^{-40} . We set $M = \lceil m/\log_2 m \rceil$ and choose the smallest row capacity L such that $M \Pr[\text{Bin}(m, 1/M) > L] \leq 2^{-40}$. The ratio ML/m denotes the expansion of the padded row-local representation relative to the input size. In our MPSU evaluation, the parameter m is instantiated with the maximum aggregate input size.

Correctness follows from Newton’s identities: provided that each row capacity is at least its actual occupancy, the stored power sums determine a polynomial whose roots are exactly the encoded elements. Our concrete row-capacity parameters are chosen to satisfy this condition with high probability, as shown in Table 1. The complete decoding algorithm, its correctness analysis, and the concrete row-capacity analysis are provided in Appendix B.

4 Our UNITED Framework

We now instantiate the batched conditional OIIS abstraction from Section 3 to construct our shuffle-free MPSU protocol. The high-level protocol is independent of the underlying cryptographic realization. For each party P_j , $j > 1$, the parties first compute secret shares of whether each element of X_j already appears in the prefix union $\bigcup_{i < j} X_i$. These shares are then used as the conditional-insertion bits for an OIIS invocation. Consequently, only the first occurrence of each distinct element is encoded.

More concretely, for every element $x_{j,q} \in X_j$, the prefix membership stage produces a secret-shared bit $b_{j,q}$ indicating whether $x_{j,q}$ has already appeared in one of the preceding sets. If $b_{j,q} = 0$, then P_j is the first party contributing that element, and the subsequent OIIS invocation inserts it into the protected sketch. If $b_{j,q} = 1$, the insertion is skipped. Hence, across all parties $P_2, \dots, P_n$, every element outside X_1 is encoded exactly once. We do not construct a protected sketch for X_1 , since P_1 is the designated output party and already knows these elements.

Each party P_j , $j > 1$, therefore produces a protected local sketch $\tilde{T}_j$ whose underlying plaintext state T_j encodes exactly the elements contributed for the first time by P_j . The parties then aggregate these protected local sketches without first revealing their contents. Because the underlying IIS insertion is symmetric, the resulting aggregate state is independent of the order in which the local sketches are combined and represents $(\bigcup_{j=1}^n X_j) \setminus X_1$. After reconstructing and decoding the aggregate sketch, P_1 locally unions the recovered set with X_1 to obtain the complete set union.

The HE- and MPC-based constructions differ only in how

the local IIS states are protected, aggregated, and reconstructed. In the HE-based realization, each T_j is encrypted under a joint public key, the encrypted sketches are sent to P_1 and homomorphically added, and the parties collaboratively decrypt the resulting aggregate. In the MPC-based realization, each T_j remains additively secret-shared; the parties locally aggregate their shares, rerandomize them with fresh additive shares of zero, and send the resulting shares to P_1 for reconstruction. In both cases, the final plaintext state encodes the same set, and only the protected representation and reconstruction mechanism differ.

This allows both OIIS constructions to use the same MPSU protocol. The prefix-membership step and the insertion rule are the same in both cases. They differ only in how the sketch is protected, aggregated, and reconstructed. Figure 11 shows the complete single-receiver protocol.

Multi-Query SSPPMT Realization. We realize $\mathcal{F}_{\text{mq-SSPPMT}}$ using parallel invocations of the batched secret-shared private membership test $\mathcal{F}_{\text{bSSPPMT}}$, followed by secure aggregation of the pairwise membership results. For a query $x_{j,v} \in X_j$, parties P_i and P_j , $i < j$, obtain binary shares satisfying $b_{i,j,v} \oplus b_{j,i,v} = \mathbf{1}[x_{j,v} \in X_i]$. These pairwise results are combined with a balanced binary OR tree evaluated using Beaver triples, yielding binary shares of $\mathbf{1}[x_{j,v} \in \bigcup_{i=1}^{j-1} X_i]$. The parties then use daBits [15] to convert the result into arithmetic shares required by the subsequent OIIS stage. Details are provided in Appendix D.4.

For the MPC realization, P_j first arranges its queries in the same padded row-slot layout used by the row-local OIIS, and the batched membership tests are executed over this fixed layout. Hence, the resulting membership-bit shares are produced directly in the corresponding (r, s) ordering, without requiring an additional oblivious permutation. All rows and slots, including padding positions, are processed uniformly.

Correctness and Security. Correctness follows from the prefix-membership bits and the correctness of the underlying OIIS realization. For every $x_{j,q} \in X_j$, $j > 1$, the bit $b_{j,q}$ equals zero exactly when P_j is the first party after P_1 contributing that element. Therefore, the conditional OIIS invocation inserts each distinct element in $(\bigcup_{j=1}^n X_j) \setminus X_1$ exactly once and skips all later occurrences. By symmetry of the underlying IIS, aggregating the local states $T_2, \dots, T_n$ yields a state $T = \sum_{j=2}^n T_j$ encoding exactly $(\bigcup_{j=1}^n X_j) \setminus X_1$. Decoding therefore returns $Y = (\bigcup_{j=1}^n X_j) \setminus X_1$, and P_1 outputs $X_1 \cup Y = \bigcup_{j=1}^n X_j$.

Security follows by sequential composition of the private prefix-membership protocol, the selected OIIS realization, and the protected aggregation procedure. The prefix-membership stage reveals only additive shares of the membership bits, while the OIIS realization hides the condition bits and the resulting local IIS states. In the HE realization, each local state remains encrypted under the joint public key and only the ag-

Parameters: n parties $P_1, \dots, P_n$, set size m , an $\mathcal{F}_{\text{mq-SSPMT}}$ protocol, an additive zero-sharing functionality $\mathcal{F}_{\text{ZeroShare}}$, and an OIIS realization $\Pi_{\text{OIIS}} \in \{\Pi_{\text{HE-OIIS}}, \Pi_{\text{MPC-OIIS}}\}$.

Input: Each party P_j , $j \in [n]$, holds a set $X_j = \{x_{j,1}, \dots, x_{j,m}\}$.

Protocol:

1. **Setup.** If $\Pi_{\text{OIIS}} = \Pi_{\text{HE-OIIS}}$, the parties invoke $\text{KeyGen}(1^\kappa, n)$ to obtain a joint public key and secret-key shares $(\text{pk}, \text{sk}_1, \dots, \text{sk}_n) \leftarrow \text{KeyGen}(1^\kappa, n)$, where pk is known to all parties and party P_i receives sk_i . If $\Pi_{\text{OIIS}} = \Pi_{\text{MPC-OIIS}}$, no public-key setup is required.
2. **Private prefix membership.** For every $j = 2, \dots, n$, party P_j and the preceding parties $P_1, \dots, P_{j-1}$ invoke $\mathcal{F}_{\text{mq-SSPMT}}$ on $X_1, \dots, X_{j-1}$ with batch queries $X_j = \{x_{j,1}, \dots, x_{j,m}\}$. For every $x_{j,q} \in X_j$, party P_i , $i \in [j]$, obtains an additive share $\langle b_{j,q} \rangle_i$ such that

$$b_{j,q} = \sum_{i=1}^j \langle b_{j,q} \rangle_i = \mathbf{1} \left[x_{j,q} \in \bigcup_{k < j} X_k \right].$$

Thus, $b_{j,q} = 0$ if P_j is the first party containing $x_{j,q}$, while $b_{j,q} = 1$ otherwise.

3. **Conditional OIIS encoding.** For every $j = 2, \dots, n$, the parties invoke Π_{OIIS} on X_j and the corresponding condition-bit shares $\{\langle b_{j,q} \rangle_i\}$. Let T_j denote the underlying plaintext IIS state encoding $U_j^* = \{x_{j,q} \in X_j \mid b_{j,q} = 0\}$, and let $\tilde{T}_j$ denote its protected representation. Thus, $\tilde{T}_j = \text{Enc}_{\text{pk}}(T_j)$ in the HE realization, while $\tilde{T}_j = \langle T_j \rangle$ in the MPC realization.
4. **Protected sketch aggregation.** The protected sketches $\tilde{T}_2, \dots, \tilde{T}_n$ are aggregated according to the selected realization:
 - *HE realization:* for every $j \in \{2, \dots, n\}$, party P_j holds $\tilde{T}_j = \text{Enc}_{\text{pk}}(T_j)$. Each party P_j sends $\tilde{T}_j$ to P_1 . Party P_1 homomorphically aggregates the received encrypted sketches as $\tilde{T} = \sum_{j=2}^n \tilde{T}_j = \text{Enc}_{\text{pk}}\left(\sum_{j=2}^n T_j\right)$.
 - *MPC realization:* for every j , the protected sketch $\tilde{T}_j = \langle T_j \rangle$ is additively secret-shared among the parties. Let $\langle T_j \rangle_i$ denote the share of T_j held by party P_i . Each party locally computes $\langle T \rangle_i = \sum_{j=2}^n \langle T_j \rangle_i$. The parties then invoke $\mathcal{F}_{\text{ZeroShare}}$ to obtain fresh additive zero shares $\langle 0 \rangle_1, \dots, \langle 0 \rangle_n$ satisfying $\sum_{i=1}^n \langle 0 \rangle_i = 0$. Each party P_i sends the rerandomized share $M_i = \langle T \rangle_i + \langle 0 \rangle_i$ to P_1 .
5. **Reconstruction and decoding.** The aggregated sketch is reconstructed according to the selected realization:
 - *HE realization:* the parties collaboratively decrypt $\tilde{T}$ using their secret-key shares $\text{sk}_1, \dots, \text{sk}_n$, so that P_1 obtains $T = \sum_{j=2}^n T_j$. Party P_1 then runs the standard IBLT decoder on T to obtain the encoded set Y .
 - *MPC realization:* party P_1 reconstructs $T = \sum_{i=1}^n M_i = \sum_{i=1}^n (\langle T \rangle_i + \langle 0 \rangle_i) = \sum_{j=2}^n T_j$. Party P_1 then runs Π_{Decode} on the reconstructed row-local power-sum IIS to obtain the encoded set Y .
6. **Union reconstruction.** Party P_1 outputs $X_1 \cup Y$.

Figure 11: UNITED: shuffle-free MPSU (Π_{MPSU}) protocols instantiated with either the HE-based or MPC-based OIIS realization.

aggregated sketch is collaboratively decrypted to P_1 . In the MPC realization, the local states remain additively secret-shared; the parties first aggregate their shares and rerandomize them with fresh additive shares of zero before sending them to P_1 , so that P_1 reconstructs only the aggregate state. Hence, assuming security of $\mathcal{F}_{\text{mq-SSPMT}}$, Π_{OIIS} , $\mathcal{F}_{\text{ZeroShare}}$, and the underlying HE or MPC primitives against semi-honest adversaries, the view of any allowed coalition can be simulated from its inputs and prescribed output. In particular, only P_1 learns the reconstructed union, and no additional information about which party contributed a particular union element is revealed. We defer the formal proof of Theorem 3 to Appendix D.3.

Theorem 3 (Security of Single-Receiver UNITED). *Assume that $\mathcal{F}_{\text{mq-SSPMT}}$, $\mathcal{F}_{\text{ZeroShare}}$, and the selected OIIS realization $\Pi_{\text{OIIS}} \in \{\Pi_{\text{HE-OIIS}}, \Pi_{\text{MPC-OIIS}}\}$ are correct and secure against semi-honest adversaries with a dishonest majority. Then the*

single-receiver protocol Π_{MPSU} in Figure 11 correctly realizes the MPSU functionality: party P_1 outputs $\bigcup_{j=1}^n X_j$, while no allowed coalition learns information about the honest parties' inputs beyond its own inputs and the prescribed output.

4.1 Multi-Receiver Reconstruction

The protocol in Figure 11 considers the standard single-receiver setting, where only P_1 receives the union. We also consider a *multi-receiver MPSU* setting in which every party receives $U = \bigcup_{j=1}^n X_j$. This setting allows the final reconstruction workload to be distributed across the parties rather than concentrated at a single receiver.

In the single-receiver protocol, the aggregate sketch encodes only $U \setminus X_1$, since P_1 already knows X_1 . In the multi-receiver setting, however, different parties reconstruct plain-

text shards of the aggregate sketch. To ensure that every reconstructed shard is determined only by the prescribed output U , we instead construct the aggregate sketch so that it encodes the full union. Specifically, party P_1 additionally constructs a protected sketch $\tilde{T}_1$ whose underlying plaintext state T_1 encodes X_1 . This insertion is unconditional and requires no prefix-membership test. Each P_j , $j > 1$, keeps the existing conditional sketch T_j encoding $U_j^* = \{x_{j,q} \in X_j : b_{j,q} = 0\}$. Since these sets are pairwise disjoint and their union is U , the aggregate $T = \sum_{j=1}^n T_j$ encodes exactly U .

We use a public partition $\{\mathcal{R}_1, \dots, \mathcal{R}_n\}$ of the sketch rows, where party P_q is responsible for the rows in $\mathcal{R}_q$. The HE- and MPC-based realizations use this partition in different ways.

HE-Based Distributed Aggregation. For the HE-based construction, party P_1 first constructs $\tilde{T}_1 = \text{Enc}_{\text{pk}}(T_1)$. Each party P_j , $j \in [n]$, partitions its encrypted sketch $\tilde{T}_j$ according to the public row partition. For every $q \in [n]$, party P_j sends $\tilde{T}_j|_{\mathcal{R}_q}$ to P_q . Party P_q homomorphically aggregates its assigned shards as $\tilde{T}|_{\mathcal{R}_q} = \sum_{j=1}^n \tilde{T}_j|_{\mathcal{R}_q}$.

The parties then collaboratively decrypt each aggregated shard, so that P_q obtains the plaintext rows $T|_{\mathcal{R}_q}$. After all shards are decrypted, they are combined to reconstruct the complete sketch T , which encodes exactly U and is decoded using the standard IBLT decoder. The resulting union is then provided to all parties. Figure 18 (in Appendix) gives the complete procedure.

MPC-Based Distributed Decoding. For the MPC-based construction, party P_1 constructs the row-local power-sum IIS state T_1 encoding X_1 and secret-shares it among the parties as $\langle T_1 \rangle$. The parties then locally aggregate all protected local sketches, including T_1 , so that each party P_i holds $\langle T \rangle_i = \sum_{j=1}^n \langle T_j \rangle_i$. Instead of sending the rerandomized shares of the entire sketch to P_1 as in the single-receiver protocol, each party P_i sends the rerandomized shares corresponding to $\mathcal{R}_q$ to party P_q . In particular, for every $q \in [n]$, party P_i sends $(\langle T \rangle_i + \langle 0 \rangle_i)|_{\mathcal{R}_q}$ to P_q . Party P_q reconstructs $T|_{\mathcal{R}_q}$ and runs the row-local power-sum decoder on its assigned rows to obtain Y_q . After all parties finish in parallel, the decoded sets are combined as $U = \bigcup_{q=1}^n Y_q$. Figure 17 (in Appendix) summarizes this procedure.

In both realizations, the multi-receiver aggregate sketch encodes the full union U , so each reconstructed shard $T|_{\mathcal{R}_q}$ is determined by the prescribed output and public parameters. Thus, distributed reconstruction reveals no additional information while parallelizing the final workload. The corresponding security proof is given in Appendix D.5.

4.2 Union-Size-Aware MPSU

Our MPSU framework represents the union using an IIS whose size depends on the number of distinct encoded el-

Parameters: n parties $P_1, \dots, P_n$, an $\mathcal{F}_{\text{mq-SSPMT}}$ protocol, M rows, and a public hash function $h_0 : \{0, 1\}^\ell \rightarrow [M]$.

Input: Each party P_j holds a set X_j . For every $j = 2, \dots, n$ and every $x_{j,q} \in X_j$, the private prefix-membership stage produces additive shares $\langle b_{j,q} \rangle_i$ such that $b_{j,q} = \sum_{i=1}^j \langle b_{j,q} \rangle_i = \mathbf{1}[x_{j,q} \in \bigcup_{k < j} X_k]$.

Protocol:

1. **Union-cardinality aggregation.** The parties locally aggregate their shares of the prefix-membership bits to obtain additive shares of $C' = \sum_{j=2}^n \sum_{q=1}^{|X_j|} (1 - b_{j,q})$. They securely reconstruct C' and compute $C = |X_1| + C'$.
2. **Row-occupancy aggregation.** In the multi-receiver setting, for every row $r \in [M]$, the parties securely compute

$$C_r = \sum_{x \in X_1} \mathbf{1}[h_0(x) = r] + \sum_{j=2}^n \sum_{q=1}^{|X_j|} \mathbf{1}[h_0(x_{j,q}) = r] (1 - b_{j,q}).$$

They reconstruct the row-occupancy vector $(C_1, \dots, C_M)$.

Output: The parties obtain $C = |\bigcup_{j=1}^n X_j|$. In the multi-receiver MPC setting, they additionally obtain $(C_1, \dots, C_M)$.

Figure 12: Union-cardinality and row-occupancy computation ($\Pi_{\text{MPSU-CA}}$) for union-size-aware UNITED.

ements. While this can reach nm for disjoint inputs, substantial overlap yields a much smaller union, making worst-case sizing unnecessarily expensive.

More precisely, let $C = |U|$. In the single-receiver protocol, the aggregate sketch encodes $U \setminus X_1$, so we define $C' = C - |X_1|$. The HE-based realization can therefore size its IBLT according to C' rather than the worst-case bound nm . In the multi-receiver protocol, the aggregate sketch encodes the full union U and is instead sized according to C .

Row-Specific Parameters for the MPC Realization. In the multi-receiver MPC realization, we can further reduce the representation by determining the required capacity of each row separately. For each row $r \in [M]$, define $C_r = |\{x \in U \mid h_0(x) = r\}|$. Thus, C_r is exactly the number of union elements assigned to row r , and $\sum_{r=1}^M C_r = C$.

Since every local set satisfies $X_j \subseteq U$, the number of elements of any X_j assigned to row r is at most C_r . Therefore, setting $L_r = C_r$ provides sufficient capacity both for every local row-padded representation and for the final aggregate row. We represent row r as $T[r] = (S_{r,0}, S_{r,1}, \dots, S_{r,L_r})$. The total number of field elements in the resulting sketch is $\sum_{r=1}^M (L_r + 1)$ instead of $M(L + 1)$.

The row occupancies can be obtained from the same prefix-membership outputs used to compute the union cardinality. The elements of X_1 are always included, while for every $x_{j,q} \in X_j$, $j > 1$, $1 - b_{j,q}$ equals one exactly when $x_{j,q}$ is contributed

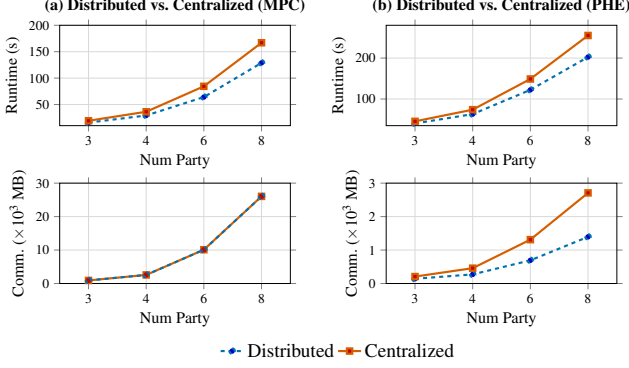


Figure 13: Effect of the use of shards distribution on MPC-based (distributed decoder) and HE-based (distributed aggregator) constructions. For each protocol, we report runtime and communication cost for $n \in \{3, 4, 6, 8\}$ and a fixed $m = 2^{16}$.

for the first time. Hence, for every row r , $C_r = \sum_{x \in X_1} \mathbf{1}[h_0(x) = r] + \sum_{j=2}^n \sum_{q=1}^{|X_j|} \mathbf{1}[h_0(x_{j,q}) = r] (1 - b_{j,q})$. Summing over all rows gives $C = \sum_{r=1}^M C_r$. Figure 12 gives the corresponding union-size computation.

Revealing the row-occupancy vector requires additional consideration beyond revealing only the union cardinality. In the multi-receiver setting, $(C_1, \dots, C_M)$ introduces no additional leakage because every party receives U and can compute the same vector from the prescribed output and the public hash function h_0 . In the single-receiver setting, these row occupancies are not determined by the output available to non-receiver parties. Therefore, the single-receiver variant reveals only C and uses a common public row capacity L , while the multi-receiver variant may reveal $(C_1, \dots, C_M)$ and use $L_r = C_r$.

Once the union-size information is available, the HE-based realization sizes its IBLT according to C' in the single-receiver setting and C in the multi-receiver setting. The MPC-based realization may additionally use the row-specific capacities $L_r = C_r$ in the multi-receiver setting. The complete union-size-aware protocol is given in Appendix D, with its correctness and security analysis in Appendix D.1. As shown in Section 5, smaller unions reduce both runtime and communication, yielding about a $2\times$ runtime improvement for the MPC construction and up to $2.36\times$ for the HE-based construction.

5 Experiment Results

We implemented and evaluated UNITED across different party counts, set sizes, and network conditions. All experiments use $\lambda = 40$ and $\kappa = 128$ and run on an AMD EPYC 74F3 server with 256 GB RAM. Network conditions are emulated using Linux `tc`: LAN (0.02 ms, 10 Gbps), WAN 1 (80 ms, 400 Mbps), and WAN 2 (80 ms, 50 Mbps). Unless otherwise stated, the MPC-based OIIS uses row-local IIS parameters

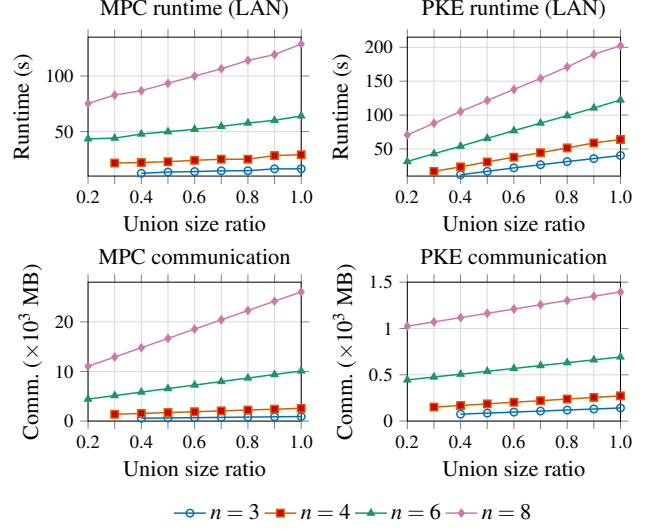


Figure 14: Effect of the union size on the performance of the multi-receiver MPC and HE-based constructions. The union size ratio is defined as the union cardinality divided by the maximum possible union size $m \times n$. We report runtime and communication cost for $n \in \{3, 4, 6, 8\}$ with fixed set size $m = 2^{16}$.

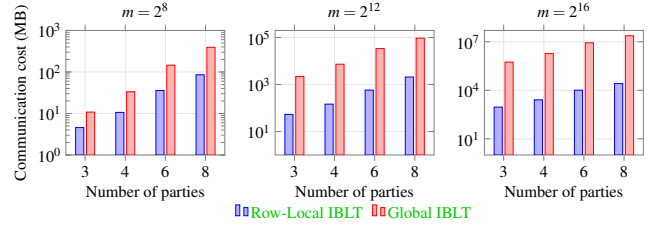


Figure 15: Communication cost of our Row-Local IIS construction compared with the naive baseline for different set sizes m and numbers of parties. The y-axis is logarithmic.

selected for the maximum aggregate input size mn , which corresponds to the parameter m in Table 1.

Distributed versus Centralized Decoder. We evaluate the benefit of distributing the final reconstruction workload for $n \in \{3, 4, 6, 8\}$ with $m = 2^{16}$. For the MPC-based construction, distributed decoding reduces runtime by up to about $1.4\times$ while maintaining comparable communication cost. For the HE-based construction, distributed aggregation and decryption reduce runtime by up to about $1.3\times$ and communication by up to about $2\times$. The complete results are shown in Figure 13.

Effect of Union Size. We evaluate the effect of union size by varying it from 10% to 100% of the maximum possible size mn , with $m = 2^{16}$ and $n \in \{3, 4, 6, 8\}$, as shown in Figure 14. For each target ratio, we generate sets with the prescribed union cardinality by sampling the union elements uniformly at

Table 2: Runtime comparison of MPSU protocols. For [4], PK and SK denote the public-key and secret-key variants, respectively. MPC-A and HE-A denote our multi-receiver variants, while MPC-1 and HE-1 denote our single-receiver variants. The best runtime overall (blue) and on single-receiver (gray, if they are different from the best overall) is highlighted. We cap the runtime at 1 hour and denote configurations with runtime more than 1 hour as >1h. OOM indicates configurations that get out-of-memory error.

m	n	LAN							WAN 1				WAN 2			
		[4] PK	[4] SK	[6]	MPC-A	HE-A	MPC-1	HE-1	[4] PK	[6]	HE-A	HE-1	[4] PK	[6]	HE-A	HE-1
2^{14}	3	12.39	2.42	22.84	3.66	5.60	3.92	5.76	16.52	30.38	10.09	10.54	19.08	39.51	13.56	16.01
	4	22.60	3.97	29.72	5.89	9.07	6.11	9.85	31.85	39.34	15.65	16.24	38.03	59.26	23.62	25.90
	6	51.96	10.29	44.33	10.28	18.88	10.46	20.59	70.04	59.81	30.78	33.62	89.39	98.85	51.44	57.85
	8	91.26	24.44	62.54	17.04	36.00	17.44	35.59	119.79	87.39	49.33	51.67	162.00	156.87	91.36	107.30
	10	140.50	33.18	85.29	27.65	42.63	30.74	49.20	178.24	112.87	63.36	72.02	258.14	210.26	142.98	170.91
2^{16}	3	45.98	7.94	88.29	11.82	18.85	12.08	20.51	56.03	100.57	28.84	30.09	64.72	135.23	45.75	50.45
	4	87.76	12.95	112.89	18.64	32.70	19.82	37.60	106.45	133.87	46.41	48.79	132.19	196.65	81.02	92.22
	6	204.98	41.68	171.05	34.80	66.67	37.33	73.44	244.52	207.83	92.59	103.70	326.43	332.96	184.76	210.69
	8	361.66	107.23	240.56	63.75	130.89	69.14	130.85	465.54	312.67	163.04	216.83	585.02	508.57	335.77	390.40
	10	561.48	140.85	324.92	102.45	165.56	114.57	181.04	639.39	434.39	221.82	257.68	901.45	734.97	502.42	613.18
2^{18}	3	188.82	33.18	364.58	30.09	70.72	32.43	80.30	191.26	375.66	94.42	100.67	243.75	501.04	173.66	189.20
	4	358.04	94.29	447.75	49.10	119.06	53.49	133.26	379.93	497.74	156.47	176.17	496.95	714.51	303.05	345.74
	6	819.91	165.01	688.43	87.20	254.92	99.69	202.43	903.88	784.62	321.10	373.38	1146	1257	659.29	752.31
	8	1462	458.96	973.00	164.62	535.42	181.32	526.65	1595	1143	623.36	661.78	2020	1951	1278	1464
	10	2293	OOM	1372	266.72	647.63	331.06	743.62	2501	1538	834.91	960.49	3071	2968	1967	2383
2^{20}	3	741.92	123.86	1376	119.23	282.27	121.49	287.93	762.18	1468	344.97	385.40	966.66	2110	652.31	691.19
	4	1435	404.61	1771	182.09	482.00	205.18	529.77	1512	1883	553.69	635.97	1901	2879	1172	1311
	6	>1h	743.20	2735	370.14	1010	415.06	1123	>1h	2987	1238	1372	>1h	>1h	2617	3046
	8	>1h	2065	>1h	665.37	1996	1184	2096	>1h	>1h	2416	2588	>1h	>1h	>1h	>1h
	10	>1h	>1h	>1h	1184	2588	1340	2957	>1h	>1h	>1h	>1h	>1h	>1h	>1h	>1h

random. The HE-based construction sizes its IBLT according to the number of elements represented by the aggregate sketch. For the multi-receiver MPC construction, we compute $C_r = |\{x \in U : h_0(x) = r\}|$ under the public hash function h_0 and set the row-specific capacity to $L_r = C_r$. As the union size decreases, both runtime and communication decrease. At the smallest evaluated ratio, runtime is reduced by about $2\times$ for the MPC-based construction and by up to $2.36\times$ for the HE-based construction.

Effect of Row-Local Design. In this experiment we compare the communication cost between the naive, direct application of IBLT solution in HE design to the MPC design with our row-local design that helps saving cost (we use Global IBLT in the Figure to refer to this baseline). We run the experiments across multiple values of m and n . The results are shown in Figure 15. The results illustrate a saving of upto more than $900\times$ in the case of $m = 2^{16}$ and $n = 8$.

Comparison with Prior Work. We compare the runtime and communication cost of UNITED with the state-of-the-art single-receiver MPSU protocols of [4, 6] using their public repositories.¹² For a fair comparison, we configure UNITED using the maximum union size $m \times n$, so that the evaluation does not rely on savings from smaller unions. Runtime results are reported in Table 2, while communication costs are

reported in Table 3 in Appendix.

Overall, UNITED achieves the best runtime in most evaluated settings, with only a few exceptions at the smallest set size $m = 2^{14}$. Ignoring runs exceeding the 1 hour cap, the MPC-based construction achieves up to a $3.10\times$ runtime improvement at $n = 8$ and $m = 2^{20}$ in LAN setting. In WAN 1, the HE-based construction achieves up to a $2.73\times$ improvement at $n = 4$ and $m = 2^{20}$, while in WAN 2 it achieves up to a $1.77\times$ improvement at $n = 6$ and $m = 2^{16}$.

6 Conclusion and Limitations

We presented UNITED, a shuffle-free MPSU framework based on Oblivious Insert-Only Invertible Sketches. By replacing element-wise message modification and multiparty shuffling with conditional insertion into a protected symmetric sketch, UNITED hides element ownership by construction. We instantiate the framework using both homomorphic encryption and additive secret sharing, and support multi-receiver reconstruction and union-size-aware parameter selection. Our evaluation shows substantial runtime improvements while reducing the round complexity from $O(n)$ to $O(\log n)$.

The remaining limitation, shared with previous state-of-the-art [4, 6], is on communication model where private membership test requires parties to communicate to all others, making UNITED not applicable to star-network architecture. Our future work will focus on solving such an issue.

¹PULSE:<https://github.com/asu-crypto/pulse>

²DCZB:<https://github.com/real-world-cryptography/MPSU>

Acknowledgement

This work was partially supported by NSF award #2451972, ARPA-H award #1AY2AX000167-01, and Amazon Research award.

References

- [1] Donald Beaver. Efficient multiparty protocols using circuit randomization. In Joan Feigenbaum, editor, *Advances in Cryptology — CRYPTO '91*, pages 420–432, Berlin, Heidelberg, 1992. Springer Berlin Heidelberg.
- [2] Marina Blanton and Everaldo Aguiar. Private and oblivious set and multiset operations. In *Proceedings of the 7th ACM Symposium on Information, Computer and Communications Security*, ASIACCS '12, pages 40–51. ACM, 2012.
- [3] Keith Bonawitz, Vladimir Ivanov, Ben Kreuter, Antonio Marcedone, H. Brendan McMahan, Sarvar Patel, Daniel Ramage, Aaron Segal, and Karn Seth. Practical secure aggregation for federated learning on user-held data, 2016. URL: <https://arxiv.org/abs/1611.04482>, [arXiv:1611.04482](https://arxiv.org/abs/1611.04482).
- [4] Minglang Dong, Cong Zhang, Yujie Bai, and Yu Chen. Efficient multi-party private set union without non-collusion assumptions. Cryptology ePrint Archive, Paper 2024/1146, 2024. URL: <https://eprint.iacr.org/2024/1146>.
- [5] Keith B. Frikken. Privacy-preserving set union. In *Applied Cryptography and Network Security*, volume 4521 of *Lecture Notes in Computer Science*, pages 237–252. Springer, 2007.
- [6] Jiahui Gao, Son Nguyen, Marina Blanton, and Ni Trieu. PULSE: Parallel private set union for large-scale entities. Cryptology ePrint Archive, Paper 2025/790, 2025. URL: <https://eprint.iacr.org/2025/790>, [doi:10.1145/3719027.3765108](https://doi.org/10.1145/3719027.3765108).
- [7] Jiahui Gao, Son Nguyen, and Ni Trieu. Toward a practical multi-party private set union. Cryptology ePrint Archive, Paper 2023/1930, 2023. URL: <https://eprint.iacr.org/2023/1930>.
- [8] Xuhui Gong, Qiang-Sheng Hua, and Hai Jin. Nearly optimal protocols for computing multi-party private set union. In *2022 IEEE/ACM 30th International Symposium on Quality of Service*. IEEE, 2022. [doi:10.1109/IWQoS54832.2022.9812897](https://doi.org/10.1109/IWQoS54832.2022.9812897).
- [9] Michael T. Goodrich and Michael Mitzenmacher. Invertible bloom lookup tables, 2015. URL: <https://arxiv.org/abs/1101.2245>, [arXiv:1101.2245](https://arxiv.org/abs/1101.2245).
- [10] Marcel Keller and Peter Scholl. Efficient, oblivious data structures for mpc. Cryptology ePrint Archive, Paper 2014/137, 2014.
- [11] Lea Kissner and Dawn Xiaodong Song. Privacy-preserving set operations. In *Advances in Cryptology – CRYPTO 2005*, volume 3621 of *Lecture Notes in Computer Science*, pages 241–257. Springer, 2005. [doi:10.1007/11535218_15](https://doi.org/10.1007/11535218_15).
- [12] Xiang Liu and Ying Gao. Scalable multi-party private set union from multi-query secret-shared private membership test. Cryptology ePrint Archive, Paper 2023/1413, 2023. URL: <https://eprint.iacr.org/2023/1413>.
- [13] Lucas Piske and Ni Trieu. Is PSI really faster than PSU? achieving efficient PSU with invertible bloom filters. Cryptology ePrint Archive, Paper 2026/376, 2026. URL: <https://eprint.iacr.org/2026/376>.
- [14] Martin Raab and Angelika Steger. “balls into bins” — a simple and tight analysis. In Michael Luby, José D. P. Rolim, and Maria Serna, editors, *Randomization and Approximation Techniques in Computer Science*, pages 159–170, Berlin, Heidelberg, 1998. Springer Berlin Heidelberg.
- [15] Dragos Rotaru and Tim Wood. MArBled circuits: Mixing arithmetic and boolean circuits with active security. Cryptology ePrint Archive, Paper 2019/207, 2019. URL: <https://eprint.iacr.org/2019/207>.
- [16] Wei Shi, Huaxiong Wang, and Josef Pieprzyk. Efficient privacy-preserving set operations. In *Information Security and Cryptology*, Lecture Notes in Computer Science. Springer, 2012.
- [17] Katsunari Shishido and Atsuko Miyaji. Efficient and quasi-accurate multiparty private set union. In *2018 IEEE International Conference on Smart Computing*, pages 309–314. IEEE, 2018.
- [18] Jelle Vos, Mauro Conti, and Zekeriya Erkin. Fast multi-party private set operations in the star topology from secure ands and ors. *Cryptology ePrint Archive*, (2022/721), 2022.
- [19] Xiao Shaun Wang, Kartik Nayak, Chang Liu, T.-H. Hubert Chan, Elaine Shi, Emil Stefanov, and Yan Huang. Oblivious data structures. Cryptology ePrint Archive, Paper 2014/185, 2014.

A Related Work

Early MPSU protocols. Early MPSU constructions relied mainly on generic MPC or public-key cryptography. Kissner

and Song [11] proposed one of the first MPSU protocols by representing each input set as a polynomial whose roots are the set elements and combining the encrypted polynomials using additive homomorphic encryption. While this approach provides a compact algebraic representation of the union, it requires a large number of homomorphic operations and high-degree polynomial computations. Frikken [5] reduced the polynomial degree and improved the efficiency of this approach, but the protocol still requires a quadratic number of public-key operations in the set size. Subsequent work explored alternative representations. Sang et al. [16] used rational polynomial functions and reversed Laurent series, while Bloom-filter-based approaches [8, 17] let each party encode its set into a local Bloom filter and homomorphically combine the resulting filters. These approaches avoid explicit processing of individual union elements, but their dependence on expensive public-key operations or false-positive-prone data structures limits their practical efficiency.

A different direction is to compute the union entirely inside generic MPC. Blanton and Aguiar [2] secret-share all input elements, obviously sort the combined inputs, and remove duplicates by comparing adjacent entries. Although this avoids expensive homomorphic polynomial evaluation, the oblivious sorting network introduces substantial computation and communication for large sets. Vos et al. [18] instead represent each set as a bit vector over the input universe and securely compute the bitwise OR of these vectors. This approach is attractive for small universes but becomes impractical when elements come from a large domain, since its cost depends on the universe size rather than only on the number of input elements.

Recent practical MPSU protocols. Recent practical MPSU protocols [4, 6, 7, 12] depart from generic MPC and heavy homomorphic evaluation and instead build on specialized two-party primitives such as SSPMTs, oblivious transfer, and multi-party shuffling. These protocols exploit the decomposition

$$X_1 \cup (X_2 \setminus X_1) \cup \dots \cup \left(X_n \setminus \bigcup_{i=1}^{n-1} X_i \right),$$

so that each party P_j contributes only those elements that have not appeared in any preceding set. For each $x \in X_j$, party P_j therefore performs pairwise membership tests with the preceding parties to determine whether $x \in \bigcup_{i < j} X_i$. The resulting membership bits remain secret-shared and are subsequently used to suppress duplicate contributions.

Liu and Gao [12] introduced an efficient symmetric-key MPSU construction based on multi-query SSPMT and random oblivious transfer. For each element, the parties obtain shares that reconstruct either to the original element when it is a first occurrence or to a random value otherwise. Before reconstruction, all resulting shares must be passed through

a multi-party secret-shared shuffle so that the leader cannot associate a surviving element with the party that contributed it. This construction substantially improved practical performance over earlier MPSU protocols, but relies on a non-collusion assumption between the leader and the remaining parties.

Gao et al. [7] developed a public-key-based alternative using a multi-key cryptosystem and membership oblivious transfer (mOT). Instead of creating secret shares of candidate elements, each element is maintained as a ciphertext. Pairwise membership tests and mOT are used to replace duplicates with encryptions of dummy values while retaining encryptions of elements that appear for the first time. The resulting ciphertexts are then processed by a collaborative shuffle-and-decrypt procedure in which parties sequentially partially decrypt, rerandomize, and permute the ciphertext collection. This prevents the leader from learning which party contributed a reconstructed element and provides security against semi-honest collusions.

Dong et al. [4] further improved this framework in both efficiency and security. A key contribution is the batched SSPMT construction, which combines hashing-to-bins with batched OPPrF and secret-shared equality testing to reduce the dependence on generic secure computation. Their symmetric-key construction introduces multi-party secret-shared random oblivious transfer to achieve standard semi-honest security against arbitrary collusion, while their public-key construction combines batched SSPMT with multi-key rerandomizable public-key encryption. The latter achieves linear computation and communication complexity in the total input size. Nevertheless, both constructions retain the same element-wise structure: each party produces protected representations of its first-occurrence elements, after which a multi-party shuffle is required to hide the correspondence between reconstructed elements and their source parties.

PULSE [6] subsequently revisited these practical protocols and abstracted them into three common modules: pairwise SSPMT, message modification, and multi-party shuffle. PULSE focuses on improving scalability with respect to the number of parties by parallelizing the expensive operations in the latter two stages. In particular, it reduces the message-modification and decryption components to constant-round computation and performs the computationally expensive parts of decryption in parallel. However, the permutation step of the multi-party shuffle remains inherently sequential in their construction, and the overall shuffle still requires $O(n)$ communication rounds.

In contrast, UNITED removes the element-wise message-modification and shuffling stages altogether. Rather than protecting and later anonymizing individual candidate elements, the parties conditionally insert first-occurrence elements into protected local sketches and then aggregate these sketches into a symmetric representation. Since the aggregate sketch does not preserve the correspondence between elements and their

Table 3: Communication comparison of MPSU protocols, measured in MB for the designated leader party P_1 . Communication cost of configurations where runtime of LAN setting more than 1 hour is not recorded.

m	n	[4] PK	[4] SK	[6]	MPC-A	HE-A	MPC-1	HE-1
2^{14}	3	16.25	289.95	56.19	159.85	31.15	218.10	39.40
	4	24.71	449.69	92.14	411.67	59.29	568.76	77.85
	6	41.29	1590	179.76	1492	149.26	2031	200.82
	8	57.87	4642	288.33	3679	302.63	4920	403.69
	10	74.41	6190	417.86	7521	498.75	9884	541.98
2^{16}	3	65.41	1296	184.41	661.62	120.39	901.93	153.39
	4	101.57	2003	301.37	1688	230.21	2332	304.46
	6	173.26	7015	584.79	6146	586.69	8388	792.95
	8	245.00	20448	934.20	15461	1198	20644	1602
	10	275.53	27289	1664	30958	2155	40649	2823
2^{18}	3	239.79	5744	891.10	1897	479.11	2580	611.11
	4	360.18	17951	1462	4842	918.44	6686	1215
	6	601.93	31038	2856	17636	2346	24034	3171
	8	843.84	90034	4586	44265	4799	59239	6416
	10	1085	—	6650	88644	8643	116639	11316
2^{20}	3	961.06	25237	3564	7849	1920	10339	2448
	4	1444	73643	5849	20062	3683	27672	4871
	6	—	135477	11425	73104	9434	99471	12734
	8	—	391836	—	183318	19304	245002	25773
	10	—	—	—	367137	34773	482671	45465

contributing parties, element-source privacy follows from the representation itself rather than from a subsequent multiparty shuffle.

Invertible sketches in private set computation. IBLTs, introduced in [9], provide compact linear representations that support additive insertion and peeling-based recovery of encoded elements. Their ability to recover all represented elements makes them useful in set reconciliation and private set computation. In the two-party setting, [13] uses an IBLT to realize PSU through interactive peeling between the parties. Directly extending this approach to MPSU, however, is problematic: if every party independently inserts its elements into an aggregate IBLT, an element held by multiple parties is inserted multiple times. The resulting multiplicity can reveal how many parties hold an element and therefore disclose intersection information beyond the union.

UNITED addresses this issue through conditional insertion. Before updating the sketch, the parties determine whether an element has already appeared in the prefix union and insert it only if it is the first occurrence. Consequently, every distinct union element appears exactly once in the aggregate sketch, independent of how many parties hold it. We capture this operation through the OIIS abstraction. Our HE-based realization applies conditional insertion to a standard IBLT, while our MPC-based realization introduces a row-local power-sum IIS that enables efficient conditional updates over additive secret shares.

Parameters: Set holders $P_1, \dots, P_{j-1}$, query party P_j , and batch size m .

Input:

- Each party P_i , $i \in [1, j-1]$, holds a set X_i .
- Party P_j holds a batch of queries $X_j = \{x_{j,1}, \dots, x_{j,m}\}$.

Protocol:

1. **Pairwise secret-shared membership tests.** For every $i \in [1, j-1]$, parties P_i and P_j invoke Π_{bSSPMT} in parallel. For every query $x_{j,v}$, party P_i receives $b_{i,j,v}$ and party P_j receives $b_{j,i,v}$ such that $b_{i,j,v} \oplus b_{j,i,v} = \mathbf{1}[x_{j,v} \in X_i]$.
2. **Binary MPC aggregation.** For every $v \in [m]$, the parties evaluate a balanced binary OR tree over the $j-1$ pairwise membership results and obtain binary shares $\beta_{1,j,v}, \dots, \beta_{j,j,v}$ satisfying

$$\bigoplus_{i=1}^j \beta_{i,j,v} = \bigvee_{i=1}^{j-1} (b_{i,j,v} \oplus b_{j,i,v}).$$

Each OR gate is evaluated as $a \vee b = a \oplus b \oplus (a \wedge b)$, where the AND operation is securely computed using Beaver triples.

3. **Binary-to-arithmetic share conversion.** Using daBits [15], the parties convert the binary shares $\beta_{1,j,v}, \dots, \beta_{j,j,v}$ into arithmetic shares $b_{1,v}, \dots, b_{j,v}$ such that

$$\sum_{i=1}^j b_{i,v} = \bigoplus_{i=1}^j \beta_{i,j,v} = \mathbf{1} \left[x_{j,v} \in \bigcup_{i=1}^{j-1} X_i \right].$$

Output: $\forall v \in [m]$, party $P_{i \in [1,j]}$, outputs the arithmetic share $b_{i,v}$.

Figure 16: Multi-query secret-shared private prefix membership test protocol $\Pi_{\text{mq-SSPMT}}$.

B Row-Local Power-Sum IIS Decoding

We provide the complete decoding procedure for the row-local power-sum IIS in Algorithm 1, its correctness analysis, and the concrete selection of the row capacity L .

Correctness. Fix a row r containing distinct elements $x_1, \dots, x_{d_r}$, and assume that $d_r \leq L$ and $p > L$, so that the divisions by $1, \dots, d_r$ in Newton’s identities are well defined in $\mathbb{F}_p$. By construction, the row stores

$$S_{r,0} = d_r \quad \text{and} \quad S_{r,j} = \sum_{\ell=1}^{d_r} x_{\ell}^j, \quad j \in [1, L+1].$$

Thus, the decoder has the first d_r power sums of the elements in the row.

Newton’s identities uniquely determine the elementary symmetric polynomials $e_{r,1}, \dots, e_{r,d_r}$ from these power sums.

Protocol $\Pi_{\text{Dist-Decode}}$: Distributed Decoding for MPC-Based UNITED

Parameters: n parties $P_1, \dots, P_n$, protected local sketches $\langle T_1 \rangle, \dots, \langle T_n \rangle$ whose aggregate encodes the full union, a public partition $\{\mathcal{R}_1, \dots, \mathcal{R}_n\}$ of the M row indices, and an additive zero-sharing functionality $\mathcal{F}_{\text{ZeroShare}}$.

Protocol:

4. **Protected-sketch aggregation and reconstruction.** Each party P_i locally computes

$$\langle T \rangle_i = \sum_{j=1}^n \langle T_j \rangle_i.$$

The parties invoke $\mathcal{F}_{\text{ZeroShare}}$ to obtain fresh additive zero shares $\langle 0 \rangle_1, \dots, \langle 0 \rangle_n$ satisfying $\sum_{i=1}^n \langle 0 \rangle_i = 0$. For every $q \in [n]$, each party P_i sends

$$M_{i,q} = (\langle T \rangle_i + \langle 0 \rangle_i)|_{\mathcal{R}_q}$$

to party P_q .

5. **Distributed decoding.** For every $q \in [n]$, party P_q reconstructs its assigned rows as

$$T|_{\mathcal{R}_q} = \sum_{i=1}^n M_{i,q},$$

and runs Π_{Decode} on the rows in $\mathcal{R}_q$ to obtain Y_q . After all parties finish in parallel, the decoded sets are combined as

$$U = \bigcup_{q=1}^n Y_q.$$

Output: Every party outputs $U = \bigcup_{j=1}^n X_j$.

Figure 17: Distributed reconstruction and decoding for the MPC-based multi-receiver UNITED realization.

In particular,

$$e_{r,k} = \frac{1}{k} \sum_{j=1}^k (-1)^{j-1} e_{r,k-j} S_{r,j}$$

for every $k \in [1, d_r + 1]$. Therefore, the polynomial constructed by the decoder satisfies

$$\begin{aligned} f_r(Z) &= Z^{d_r} - e_{r,1} Z^{d_r-1} + e_{r,2} Z^{d_r-2} + \dots + (-1)^{d_r} e_{r,d_r} \\ &= \prod_{\ell=1}^{d_r} (Z - x_\ell). \end{aligned}$$

Hence, the roots of f_r are exactly the elements encoded in row r .

Since the public hash function h_0 assigns every encoded element to exactly one row, the row sets are disjoint with respect to their placement in the sketch. Therefore, if every

Protocol $\Pi_{\text{Dist-Agg}}$: Distributed Aggregation for HE-Based UNITED

Parameters: n parties $P_1, \dots, P_n$, protected local sketches $\tilde{T}_1, \dots, \tilde{T}_n$ whose aggregate encodes the full union, a public partition $\{\mathcal{R}_1, \dots, \mathcal{R}_n\}$ of the sketch rows, and a multi-key partially homomorphic encryption scheme with joint public key pk and secret-key shares $\text{sk}_1, \dots, \text{sk}_n$.

Protocol:

4. **Distributed protected-sketch aggregation.** For every $j \in [n]$ and every $q \in [n]$, party P_j sends the encrypted shard $\tilde{T}_j|_{\mathcal{R}_q}$ to party P_q . Each party P_q homomorphically aggregates the received shards as

$$\tilde{T}|_{\mathcal{R}_q} = \sum_{j=1}^n \tilde{T}_j|_{\mathcal{R}_q}.$$

5. **Distributed decryption and decoding.** For every $q \in [n]$, the parties collaboratively decrypt $\tilde{T}|_{\mathcal{R}_q}$ using their secret-key shares, so that party P_q obtains $T|_{\mathcal{R}_q}$. The plaintext shards are combined to reconstruct

$$T = \bigcup_{q=1}^n T|_{\mathcal{R}_q}.$$

The parties run the standard IBLT decoder on T to recover

$$U = \bigcup_{j=1}^n X_j.$$

Output: Every party outputs $U = \bigcup_{j=1}^n X_j$.

Figure 18: Distributed aggregation and decryption for the HE-based multi-receiver UNITED realization.

row satisfies $d_r \leq L$, Algorithm 1 recovers every encoded element and no additional elements. Taking the union of the decoded row sets consequently returns exactly the set encoded in T .

The only probabilistic source of decoding failure in this construction is therefore row overflow: if some row contains more than L elements, then the sketch does not store enough power sums to reconstruct that row. We select L so that this event occurs with probability at most the target statistical error.

Concrete Row-capacity Selection. Rather than relying only on an asymptotic maximum-load bound, we select the padded row capacity L using the finite-size occupancy distribution. Let $M = \lceil m / \log_2 m \rceil$ and assume that the public hash function h_0 assigns each item independently and uniformly to one of the M rows. Then the occupancy L_r of any fixed row follows $L_r \sim \text{Bin}(m, 1/M)$. For a row capacity L , a union

Algorithm 1 Π_{Decode} : Row-Local Power-Sum IIS Decoding

Require: A field $\mathbb{F}_p$, M rows, and a reconstructed row-local power-sum IIS $T = \{T[r]\}_{r \in [M]}$, where $T[r] = (S_{r,0}, S_{r,1}, \dots, S_{r,L})$.

Ensure: The set Y encoded in T .

```
1: Let  $\text{Roots}_{\mathbb{F}_p}(f)$  denote a polynomial root-finding algorithm that returns  $\{x \in \mathbb{F}_p \mid f(x) = 0\}$ .
2:  $Y \leftarrow \emptyset$ 
3: for  $r = 1, \dots, M$  do
4:    $d_r \leftarrow S_{r,0}$ 
5:    $e_{r,0} \leftarrow 1$ 
6:   for  $k = 1, \dots, d_r$  do
7:      $e_{r,k} \leftarrow \frac{1}{k} \sum_{j=1}^k (-1)^{j-1} e_{r,k-j} S_{r,j}$ 
8:   end for
9:   Construct  $f_r(Z) \leftarrow Z^{d_r} - e_{r,1}Z^{d_r-1} + e_{r,2}Z^{d_r-2} + \dots + (-1)^{d_r} e_{r,d_r}$ .
10:   $Y_r \leftarrow \text{Roots}_{\mathbb{F}_p}(f_r)$ 
11:   $Y \leftarrow Y \cup Y_r$ 
12: end for
13: return  $Y$ 
```

bound over all rows gives

$$\Pr \left[\max_{r \in [M]} L_r > L \right] \leq M \Pr \left[\text{Bin} \left(m, \frac{1}{M} \right) > L \right].$$

Thus, the probability that any such layout contains an overflowing row is bounded by

$$\epsilon_{\text{row}}(m, n, L) = M \Pr \left[\text{Bin} \left(m, \frac{1}{M} \right) > L \right].$$

We choose the smallest L such that $\epsilon_{\text{row}}(m, n, L) \leq 2^{-\lambda}$. Table 1 reports the resulting capacities for several values of m for $< 2^{-40}$ overflow rate. The ratio $(ML)/m$ captures the amount of padded row-slot work relative to the original set size and therefore reflects the number of SSPMT instances required by the construction. As shown in Table 1, the selected row capacities keep the total padded row-slot overhead small across all evaluated parameter settings.

C Preliminaries

C.1 Multi-Key Cryptosystem

For the public-key-based realization, we follow prior MPSU work and use a multi-key cryptosystem supporting partial homomorphic evaluation. We adopt the definition of a multi-key cryptosystem from [6], as follows:

Definition 3 (Multi-key Cryptosystem). *A Multi-key Cryptosystem [6] is defined as a tuple of algorithm $(\text{KeyGen}, \text{Enc}, \text{ParDec}, \text{FinalDec}, \text{ReRand})$ specified as follow:*
Key Generation: $(pk, sk_1, \dots, sk_n) \leftarrow \text{KeyGen}(1^\kappa, n)$. The

key generation algorithm takes as input a security parameter κ and the number of parties n and outputs a joint public key pk to everyone and a secret key sk_i to each party P_i .

Encryption: $ct \leftarrow \text{Enc}(pk, m)$. Given a joint public key pk and a message m from the message space $\mathcal{M}$, the encryption algorithm computes a ciphertext ct .

Partial decryption: $ct' \leftarrow \text{ParDec}(sk_i, ct, A)$. The partial decryption algorithm takes a secret key sk_i , a ciphertext ct from the ciphertext space $\mathcal{C}$, and a set of indices $A \subseteq [n]$ such that $i \in A$. The ciphertext is interpreted as being encrypted under the partial public key $pk_A = \text{Combine}(\{sk_j \mid j \in A\})$ and the algorithm outputs another ciphertext $ct' \leftarrow \mathcal{C}$ encrypting the same message under the partial public key $pk_{A \setminus \{i\}} = \text{Combine}(\{sk_j \mid j \in A, j \neq i\})$.

Full decryption: $m \leftarrow \text{FinalDec}(sk_1, sk_2, \dots, sk_n; ct)$. The full decryption algorithm takes a ciphertext $ct \leftarrow \mathcal{C}$ encrypted under pk and all of the secret keys and outputs a message $m \leftarrow \mathcal{M}$.

In addition to $(\text{KeyGen}, \text{Enc}, \text{ParDec}, \text{FinalDec})$ functions, in this work we also consider the following additional property for the multi-key cryptosystem:

Partial Homomorphic Encryption We consider a function Eval which performs homomorphic addition of two ciphertexts (or a plaintext and a ciphertext) to a new ciphertext encrypting the sum of the two corresponding plaintexts, and homomorphic multiplication between a ciphertext and a plaintext to a new ciphertext encrypting the product. Formally, given $ct_i = \text{Enc}(pk, m_i)$ for $i \in \{1, 2\}$, the Eval function satisfies the following 3 equations:

$$\text{FinalDec}(sk_1, \dots, sk_n; \text{Eval}(+, ct_1, ct_2)) = m_1 + m_2,$$

$$\text{FinalDec}(sk_1, \dots, sk_n; \text{Eval}(+, m_1, ct_2)) = m_1 + m_2,$$

$$\text{FinalDec}(sk_1, \dots, sk_n; \text{Eval}(*, ct_1, m_2)) = m_1 * m_2$$

We refer to [6] for detailed construction of a such a multikey cryptosystem using EC-ElGamal encryption scheme.

In this paper, for shake of readability, we use phases such as “evaluate the formula homomorphically” or “evaluate the formula in the ciphertext space” instead of actual formula of Eval .

C.2 Invertible Bloom Lookup Tables

We briefly review the IBLT algorithms used in this work. An IBLT is parameterized by a set of hash functions that map each element to a small number of bins. Each bin stores the metadata required for insertion, deletion, and peeling-based recovery. We write prm for the public IBLT parameters.

Encoding. Given a set X , the encoding algorithm initializes an empty IBLT and inserts every element $x \in X$ into the bins selected by the hash functions. We denote the resulting table by

$$\text{IBLT} \leftarrow \text{Encode}_{\text{prm}}(X).$$

Since each element updates only a constant number of bins, encoding runs in time linear in $|X|$.

Deletion. Given an IBLT and a set X , the deletion operation reverses the updates corresponding to each element of X :

$$\text{IBLT}' \leftarrow \text{DeleteSet}_{\text{prm}}(\text{IBLT}, X).$$

As with insertion, deletion is linear in the number of processed elements.

Listing. The listing algorithm

$$Y \leftarrow \text{List}_{\text{prm}}(\text{IBLT})$$

recovers the encoded elements using iterative peeling. At each step, the decoder identifies a bin containing a single recoverable element, outputs that element, and removes its contribution from the other bins addressed by the same hash functions. The process continues until no peelable bins remain. If all inserted elements are recovered, the algorithm outputs the decoded set; otherwise, it returns failure.

The decoding success probability depends on the table size, the number of hash functions, and the number of encoded elements. When the table is sized above the corresponding decoding threshold, the probability of failure is negligible [9]. In our implementation, we adopt the parameter choices of [13].

For the HE-based realization of OIIS, we use only the insert-only functionality of the IBLT during sketch construction. Deletion and listing are required only for the standard IBLT interface and final decoding.

C.3 Batch SSPMT Realization

In [4], batch SSPMT is realized by having receiver Q encode their set into a Cuckoo Hash table while sender S encode their set into simple hash table and encode an OKVS for each row of the table to value s_r . SSPMT is done by performing private equality check between s_r and $\text{OKVS.Decode}(x_r)$ where x_r being the item at row r of the Cuckoo Hash of Q .

In our protocol, we cannot do the same: each row of Q 's table consists of T items (including padded items) rather than single item in the Cuckoo hash table. If we apply the same decode and equality check protocol, Q can infer extra information such as whether two items x and y in the row both belong to S by checking $\text{OKVS.Decode}(x) == \text{OKVS.Decode}(y)$.

Hence, we modify the algorithm slightly: For each row, S construct a smaller simple hash table and Q construct a row-local cuckoo hash table. Then, S create OKVS for the

row-local simple hash table while Q evaluate on its row-local cuckoo hash table. This makes sure each item in the row of Q evaluates to different values and let Q learns no extra information about membership of its value. We notice that the private equality check still returns a secret share of the membership test bit to S and Q , and the parties use such secret share to update the table using MPC and the row local $e_r(x)$ as detailed in Section 3.2.2.

D Union-Size-Aware UNITED Protocol

Figure 19 gives the complete union-size-aware version of UNITED. The protocol differs from Figure 11 only in that the parties first reconstruct the union cardinality and use it to select the parameters of the underlying IIS before running the conditional OIIS encoding stage.

D.1 Correctness and Security of Union-Size-Aware UNITED

Correctness follows from the fact that the revealed cardinality C and, in the multi-receiver setting, the row occupancies $(C_1, \dots, C_M)$ determine valid parameters for the underlying IIS. Security follows because C is explicitly revealed to all parties, while $(C_1, \dots, C_M)$ is a deterministic function of the prescribed union U and the public hash function h_0 .

Formally, let $U = \bigcup_{j=1}^n X_j$, $C = |U|$, and $C' = |U \setminus X_1| = C - |X_1|$. Recall that for every $j > 1$ and every $x_{j,q} \in X_j$, the prefix-membership stage produces additive shares of $b_{j,q} = \mathbf{1}[x_{j,q} \in \bigcup_{k < j} X_k]$. Hence, $1 - b_{j,q} = 1$ exactly when P_j is the first party containing $x_{j,q}$.

Theorem 4 (Correctness and Security of Union-Size-Aware UNITED). *Assume that the underlying UNITED protocol and its building blocks are correct and secure against semi-honest adversaries with a dishonest majority. Then the union-size-aware variant correctly and securely realizes the corresponding MPSU functionality augmented with the union cardinality C . In the multi-receiver setting, the parties may additionally use the full-union row occupancies $C_r = |\{x \in U : h_0(x) = r\}|$, $r \in [M]$, without additional leakage beyond the prescribed union output.*

Proof. We prove correctness and security separately.

Correctness of the cardinality computation. Consider an arbitrary element $x \in U$, and let $j^* = \min\{j : x \in X_j\}$ be the index of the first party containing x . If $j^* = 1$, then every later occurrence has $b_{j,q} = 1$ and contributes zero to $\sum_{j=2}^n \sum_{q=1}^{|X_j|} (1 - b_{j,q})$. If $j^* > 1$, its first occurrence has $b_{j^*,q} = 0$ and contributes one, while every later occurrence contributes zero. Thus, $C' = \sum_{j=2}^n \sum_{q=1}^{|X_j|} (1 - b_{j,q}) = |U \setminus X_1|$. Since $X_1 \subseteq U$, we obtain $C = |X_1| + C' = |U|$.

Protocol $\Pi_{\text{UA-MPSU}}$: Union-Size-Aware UNITED

Parameters: n parties $P_1, \dots, P_n$, an $\mathcal{F}_{\text{mq-SSPMT}}$ protocol, an additive zero-sharing functionality $\mathcal{F}_{\text{ZeroShare}}$, and an OIIS realization $\Pi_{\text{OIIS}} \in \{\Pi_{\text{HE-OIIS}}, \Pi_{\text{MPC-OIIS}}\}$.

Input: Each party P_j , $j \in [n]$, holds a set X_j .

Protocol:

1. **Setup.** If $\Pi_{\text{OIIS}} = \Pi_{\text{HE-OIIS}}$, the parties invoke $(\text{pk}, \text{sk}_1, \dots, \text{sk}_n) \leftarrow \text{KeyGen}(1^\kappa, n)$, where pk is public and party P_i receives sk_i . If $\Pi_{\text{OIIS}} = \Pi_{\text{MPC-OIIS}}$, no public-key setup is required.
2. **Private prefix membership.** For every $j = 2, \dots, n$, party P_j and the preceding parties $P_1, \dots, P_{j-1}$ invoke $\mathcal{F}_{\text{mq-SSPMT}}$ on $X_1, \dots, X_{j-1}$ with batch queries X_j . For every $x_{j,q} \in X_j$, party P_i , $i \in [j]$, obtains an additive share $\langle b_{j,q} \rangle_i$ such that $b_{j,q} = \sum_{i=1}^j \langle b_{j,q} \rangle_i = \mathbf{1}[x_{j,q} \in \bigcup_{k < j} X_k]$.
3. **Union-cardinality reconstruction.** The parties locally aggregate their shares of the prefix-membership bits to obtain additive shares of $C' = \sum_{j=2}^n \sum_{q=1}^{|X_j|} (1 - b_{j,q})$. They securely reconstruct C' and compute $C = |X_1| + C' = \left| \bigcup_{j=1}^n X_j \right|$. The value C is revealed to all parties.
4. **IIS parameter selection.** In the single-receiver setting, the HE realization sizes the IBLT according to C' , while the MPC realization uses a common public row capacity L chosen as a safe upper bound on the row occupancy. In the multi-receiver setting, the underlying IIS is sized according to C . For the MPC realization, the parties additionally reconstruct $C_r = \sum_{x \in X_1} \mathbf{1}[h_0(x) = r] + \sum_{j=2}^n \sum_{q=1}^{|X_j|} \mathbf{1}[h_0(x_{j,q}) = r](1 - b_{j,q})$ for every $r \in [M]$, and set $L_r = C_r$.
5. **Conditional OIIS encoding.** For every $j = 2, \dots, n$, the parties invoke Π_{OIIS} on X_j and the corresponding condition-bit shares. Let T_j denote the underlying plaintext IIS state encoding $U_j^* = \{x_{j,q} \in X_j \mid b_{j,q} = 0\}$, and let $\tilde{T}_j$ denote its protected representation. In the multi-receiver setting, party P_1 additionally constructs a protected sketch $\tilde{T}_1$ whose underlying state T_1 encodes X_1 unconditionally.
6. **Protected sketch aggregation.**
 - *Single receiver:* the protected sketches $\tilde{T}_2, \dots, \tilde{T}_n$ are aggregated as in Figure 11, yielding an aggregate state encoding $U \setminus X_1$.
 - *Multi receiver:* the protected sketches $\tilde{T}_1, \dots, \tilde{T}_n$ are aggregated using the distributed procedure of Section 4.1, yielding an aggregate state $T = \sum_{j=1}^n T_j$ encoding the full union U .
7. **Reconstruction and decoding.**
 - *Single receiver:* party P_1 reconstructs and decodes the aggregate sketch to obtain $Y = U \setminus X_1$, and outputs $X_1 \cup Y = U$.
 - *Multi receiver:* the parties use the distributed HE-based or MPC-based reconstruction procedure to decode the aggregate sketch, which directly yields U .

Output: The prescribed MPSU output is produced, and every party additionally obtains the union cardinality C . In the multi-receiver MPC setting, the row-occupancy vector $(C_1, \dots, C_M)$ may also be revealed.

Figure 19: Union-size-aware UNITED instantiated with either the HE-based or MPC-based OIIS realization.

Correctness of the selected IIS parameters. For every $j > 1$, define $U_j^* = \{x_{j,q} \in X_j : b_{j,q} = 0\}$. The sets $U_2^*, \dots, U_n^*$ are pairwise disjoint and satisfy $\bigcup_{j=2}^n U_j^* = U \setminus X_1$.

In the single-receiver setting, the aggregate sketch therefore encodes exactly C' distinct elements. The HE realization may size its IBLT according to C' , while the MPC realization uses a common public row capacity L chosen as a safe upper bound on row occupancy. Decoding recovers $U \setminus X_1$, and P_1 outputs U .

In the multi-receiver setting, party P_1 additionally contributes T_1 encoding X_1 . Hence, $T = \sum_{j=1}^n T_j$ encodes exactly U . For each row r , define $C_r = |\{x \in U : h_0(x) = r\}|$. Since every local set satisfies $X_j \subseteq U$, the number of elements of any local set assigned to row r is at most C_r . Thus, choosing $L_r = C_r$ provides sufficient capacity for both every local

row-padded representation and the final aggregate row. The row-local decoder therefore recovers all elements of U .

Security. The values $b_{j,q}$ remain secret-shared, and the parties reconstruct only their aggregate cardinality information. The revealed value $C = |U|$ is explicitly included in the output of the union-size-aware functionality. The selected IIS parameters are deterministic functions of this value and public protocol parameters and therefore reveal no additional information.

In the multi-receiver setting, the parties may additionally reveal $C_r = |\{x \in U : h_0(x) = r\}|$. Since every party receives U and h_0 is public, the vector $(C_1, \dots, C_M)$ can be computed directly from the prescribed output. A simulator given U can therefore reproduce the same row occupancies and all IIS

parameters derived from them.

The remaining execution is identical to the corresponding secure single- or multi-receiver UNITED protocol except for these public parameters. By sequential composition, the union-size-aware variant correctly and securely realizes the corresponding MPSU functionality augmented with C against semi-honest adversaries with a dishonest majority. $\square$

D.2 Proof of Security for MPC-Based OHS

Proof. Let P_Q be the data owner with input set X_Q , and let the participating parties be $P_1, \dots, P_Q$. For every row-slot position (r, s) , let $\hat{x}_{r,s}$ denote the corresponding real or padding element, and let $b_{r,s} \in \{0, 1\}$ denote the reconstructed condition bit. Recall that $b_{r,s} = 0$ indicates insertion and $b_{r,s} = 1$ indicates that the insertion is skipped.

Correctness. For every position (r, s) , party P_Q computes the row-local power vector $e_r(\hat{x}_{r,s})$. If $\hat{x}_{r,s}$ is a padding element, then $e_r(\hat{x}_{r,s}) = \mathbf{0}$. The parties hold additive shares of both $b_{r,s}$ and $e_r(\hat{x}_{r,s})$ and jointly compute an additive sharing of

$$\gamma_{r,s} = (1 - b_{r,s})e_r(\hat{x}_{r,s}).$$

They then update the corresponding row by adding $\gamma_{r,s}$.

If $b_{r,s} = 0$, then $\gamma_{r,s} = e_r(\hat{x}_{r,s})$, so the update is exactly a row-local IIS insertion. If $b_{r,s} = 1$, then $\gamma_{r,s} = \mathbf{0}$ and the sketch remains unchanged. Padding positions also contribute $\mathbf{0}$. Therefore, after all positions are processed, the reconstructed sketch state encodes exactly the set

$$\{\hat{x}_{r,s} \mid b_{r,s} = 0, \hat{x}_{r,s} \neq \perp\}.$$

Since row updates are additive and hence commutative, the final sketch is independent of the order in which the row-slot positions are processed. Thus, the protocol produces exactly the plaintext IIS state prescribed by the batched conditional OHS functionality.

Security. Let $\mathcal{A} \subsetneq [Q]$ be an arbitrary semi-honest coalition. We construct a simulator $\mathcal{S}$ for the view of $\mathcal{A}$ using only the corrupted parties' inputs and their prescribed output shares from the ideal OHS functionality.

The protocol manipulates three types of private values: the condition bits $b_{r,s}$, the row-local power vectors $e_r(\hat{x}_{r,s})$, and the intermediate products $\gamma_{r,s}$. All of these values remain additively secret-shared throughout the protocol. By privacy of additive secret sharing, any strict subset of the shares reveals no information about the underlying plaintext values beyond what is implied by the corrupted parties' own shares.

The only interactive computation performed on these shares is the multiplication required to obtain

$$\gamma_{r,s} = (1 - b_{r,s})e_r(\hat{x}_{r,s}).$$

By the assumed security of the arithmetic MPC protocol, the view of the corrupted coalition during each multiplication can be simulated from its input shares and output shares alone. Hence, the simulator invokes the MPC simulator independently for every row-slot position and obtains a transcript computationally indistinguishable from the corresponding real execution.

The subsequent update

$$\langle T_Q[r] \rangle \leftarrow \langle T_Q[r] \rangle + \langle \gamma_{r,s} \rangle$$

is performed locally and requires no communication. Therefore, it contributes no additional transcript to simulate. Since addition on additive shares is local, the simulator can maintain simulated sketch shares consistently throughout the execution.

We next consider the row-slot layout. The data owner P_Q knows which positions contain real elements, but the other parties receive only secret shares of the corresponding power vectors and condition bits. Every row-slot position is processed using the same MPC computation. Consequently, the view of a corrupted party not holding X_Q does not reveal whether a position contains a real element, whether the corresponding condition bit equals zero or one, or whether the update changes the underlying plaintext row. The outer row index is public and does not introduce additional leakage because every slot in every row follows the same computation pattern.

Finally, the ideal functionality outputs only additive shares of the resulting sketch state. The simulator therefore samples the honest parties' shares so that, together with the corrupted parties' prescribed output shares, they reconstruct to the ideal plaintext state. By the privacy of additive sharing, the resulting simulated shares are distributed identically to the shares generated in the real protocol.

Thus, for every semi-honest coalition $\mathcal{A} \subsetneq [Q]$, there exists a PPT simulator $\mathcal{S}$ such that

$$\text{View}_{\mathcal{A}}^{\text{real}} \approx_c \text{View}_{\mathcal{A}}^{\text{ideal}}.$$

Therefore, $\Pi_{\text{MPC-OHS}}$ correctly and securely realizes the batched conditional OHS encoding functionality against semi-honest adversaries. $\square$

D.3 Proof of Theorem 3

We prove that the single-receiver protocol Π_{MPSU} in Figure 11 correctly and securely realizes the MPSU functionality against semi-honest adversaries, assuming the security of $\mathcal{F}_{\text{mq-SSPPMT}}$, $\mathcal{F}_{\text{ZeroShare}}$, and the selected OHS realization. We first prove correctness and then give a simulation-based security argument.

Correctness. Let $U = \bigcup_{j=1}^n X_j$ denote the final union. For every party P_j , $j > 1$, and every element $x_{j,q} \in X_j$, the private

prefix-membership stage computes additive shares of

$$b_{j,q} = \mathbf{1} \left[x_{j,q} \in \bigcup_{k < j} X_k \right].$$

Therefore, $b_{j,q} = 0$ exactly when $x_{j,q}$ does not occur in any preceding party's set. Equivalently, P_j is the first party in the ordering $P_1, \dots, P_n$ that contributes $x_{j,q}$. If $x_{j,q}$ already occurs in some preceding set, then $b_{j,q} = 1$.

By correctness of the conditional OIS realization, the local plaintext state T_j therefore encodes exactly

$$U_j^* = \{x_{j,q} \in X_j \mid b_{j,q} = 0\}.$$

Consider any element $x \in U \setminus X_1$. Let $j > 1$ be the smallest index such that $x \in X_j$. For this occurrence, $x \notin \bigcup_{k < j} X_k$, and hence $b_{j,q} = 0$. Thus, x is inserted into T_j . For every later occurrence of x in some $X_{j'}$ with $j' > j$, we have $x \in \bigcup_{k < j'} X_k$, and therefore the corresponding condition bit is one and the insertion is skipped. Hence, every element of $U \setminus X_1$ is encoded exactly once.

Conversely, if $x \in X_1$, then for every occurrence of x in a later party P_j , we have $x \in \bigcup_{k < j} X_k$, and therefore $b_{j,q} = 1$. Thus, no element of X_1 is encoded in any T_j , $j > 1$. It follows that

$$\bigcup_{j=2}^n U_j^* = U \setminus X_1.$$

Because the underlying IIS insertion operation is symmetric, the order in which the local states are aggregated does not affect the resulting plaintext state. Both realizations therefore reconstruct an aggregate state T corresponding to the insertion of exactly the elements in $U \setminus X_1$. In particular,

$$T = \sum_{j=2}^n T_j \equiv \text{IIS.Insert}(U \setminus X_1, 0).$$

By correctness of the underlying IIS decoder,

$$Y = \text{IIS.Decode}(T) = U \setminus X_1.$$

Finally, P_1 outputs $X_1 \cup Y$, and hence

$$X_1 \cup Y = X_1 \cup (U \setminus X_1) = U = \bigcup_{j=1}^n X_j.$$

Therefore, Π_{MPSU} is correct whenever the selected IIS parameters are sufficient for successful decoding.

Security. We prove security in the standard simulation-based sense. Let $\mathcal{A} \subseteq [n]$ be an arbitrary coalition of corrupted semi-honest parties, and let $\mathcal{H} = [n] \setminus \mathcal{A}$ denote the set of honest parties. The simulator $\mathcal{S}$ is given the inputs $\{X_i\}_{i \in \mathcal{A}}$ of the corrupted parties. If $P_1 \in \mathcal{A}$, then $\mathcal{S}$ additionally receives the prescribed MPSU output $U = \bigcup_{j=1}^n X_j$. If $P_1 \notin \mathcal{A}$, the

corrupted parties receive no output from the single-receiver MPSU functionality.

We show how $\mathcal{S}$ simulates the view of $\mathcal{A}$ throughout the protocol. The view consists of the corrupted parties' inputs, local randomness, outputs of the invoked secure functionalities, protected OIS states visible to corrupted parties, messages exchanged during aggregation, and messages exchanged during reconstruction or collaborative decryption. We consider each stage separately.

Setup. In the MPC realization, there is no public-key setup and therefore nothing additional must be simulated. In the HE realization, the protocol generates a joint public key pk and secret-key shares $\text{sk}_1, \dots, \text{sk}_n$. The simulator runs the key-generation algorithm internally and gives every corrupted party P_i the resulting public key pk and its corresponding simulated secret-key share sk_i . Since this distribution is identical to a real execution of KeyGen, the setup view is perfectly distributed as in the real protocol. Secret-key shares belonging to honest parties are retained internally by the simulator and are never revealed to $\mathcal{A}$.

Private Prefix Membership. Consider the invocation associated with party P_j and its queries X_j . For each $x_{j,q} \in X_j$, the functionality computes

$$b_{j,q} = \mathbf{1} \left[x_{j,q} \in \bigcup_{k < j} X_k \right]$$

and distributes additive shares $\langle b_{j,q} \rangle_1, \dots, \langle b_{j,q} \rangle_j$.

By the assumed security of $\mathcal{F}_{\text{mq-SSPPMT}}$, there exists a simulator for each such invocation that produces the view of the corrupted participants using only their inputs and prescribed output shares. We therefore invoke this simulator independently for every prefix-membership call.

It is useful to consider why these shares do not reveal the underlying membership result. Whenever at least one participant in the invocation is honest, the shares held by the corrupted parties do not determine $b_{j,q}$ because the missing honest share acts as an additive mask. Thus, the corrupted parties learn no information about whether $x_{j,q}$ appears in an honest preceding set beyond what follows from their own inputs and eventual prescribed output. If all participants of a particular invocation are corrupted, then the query holder P_j and all preceding set holders involved in that invocation are themselves corrupted, so the corresponding membership result is already determined by the coalition's own inputs. Hence, this case introduces no additional leakage.

By sequential composition, replacing every real $\mathcal{F}_{\text{mq-SSPPMT}}$ execution by its simulator produces a view computationally indistinguishable from the real prefix-membership stage.

Conditional OIIS Encoding. For every $j > 1$, the parties invoke Π_{OIIS} using X_j and the secret shares of the corresponding condition bits. We rely directly on the security of the selected OIIS realization. By definition, conditional OIIS hides the private condition bits, the elements contributed by honest data owners, and the modifications to the underlying IIS state, except for information explicitly available to the designated participants.

For an honest party P_j , the simulator does not know X_j unless it can be derived from the ideal output. Instead, it invokes the simulator guaranteed by the security of Π_{OIIS} to produce the corrupted parties' views of the conditional-encoding execution. The resulting protected state can therefore be simulated without revealing which elements of X_j were inserted or skipped.

For a corrupted party P_j , its set X_j is already known to $\mathcal{S}$. The simulator can therefore reproduce all computations performed locally by P_j and use the OIIS simulator for the remaining messages involving honest parties.

The security requirement is slightly different in the two realizations. In the HE realization, the protected output is an encryption $\tilde{T}_j = \text{Enc}_{\text{pk}}(T_j)$. Semantic security and rerandomization ensure that, without collaborative decryption, this ciphertext does not reveal the plaintext local state T_j . In particular, P_1 may receive the encrypted local states during the subsequent aggregation stage, but it does not learn the elements encoded in any individual T_j .

In the MPC realization, the protected output is an additive sharing $\tilde{T}_j = \langle T_j \rangle$. Any coalition missing at least one required share cannot reconstruct T_j . If all participants of an invocation are corrupted, then the data owner P_j is also corrupted, and its contribution is already part of the coalition's input. Thus, the protected local state reveals no additional information about an honest party's contribution.

It follows from the security of Π_{OIIS} that the complete conditional-encoding stage can be simulated from the coalition's prescribed information.

HE Aggregation. We now consider the HE realization. Each P_j , $j > 1$, sends $\tilde{T}_j = \text{Enc}_{\text{pk}}(T_j)$ to P_1 , and P_1 computes

$$\tilde{T} = \sum_{j=2}^n \tilde{T}_j = \text{Enc}_{\text{pk}}\left(\sum_{j=2}^n T_j\right).$$

We distinguish whether the receiver P_1 is corrupted.

Suppose first that $P_1 \notin \mathcal{A}$. The corrupted parties do not receive the encrypted local sketches of honest parties, since these messages are sent to the honest receiver. Each corrupted P_j sees only its own protected state and the messages required by the collaborative-decryption protocol. The former can be simulated from its known input and the OIIS simulator, while the latter can be simulated by the security of the multi-key decryption protocol. Since no aggregate plaintext is delivered to a corrupted party, the simulator does not need to know U .

Now suppose that $P_1 \in \mathcal{A}$. In this case, the ideal MPSU functionality gives U to the simulator. Since X_1 is also known as a corrupted input, the simulator can compute

$$Y = U \setminus X_1.$$

Using the public IIS algorithms and public sketch parameters, the simulator can construct the plaintext aggregate state

$$T = \text{IIS.Insert}(Y, 0).$$

Thus, the simulator knows exactly the plaintext state that must be obtained after collaborative decryption.

For encrypted local sketches corresponding to corrupted parties, $\mathcal{S}$ computes the same ciphertext distributions as in the real protocol, since it knows their inputs and simulated membership information. For honest parties, the simulator generates ciphertexts whose individual plaintext decomposition need not match the honest parties' real local states, provided that their aggregate is consistent with T . For example, after accounting for the local states contributed by corrupted parties, the simulator may choose arbitrary dummy states for all but one honest party and choose the final honest dummy state so that the sum of all local states equals T .

More explicitly, let $\mathcal{C} = \mathcal{A} \cap \{2, \dots, n\}$ and $\mathcal{H}' = \mathcal{H} \cap \{2, \dots, n\}$. Let $T_{\mathcal{C}} = \sum_{j \in \mathcal{C}} T_j$ denote the aggregate contribution determined by corrupted parties. If $\mathcal{H}' \neq \emptyset$, choose arbitrary dummy states $\hat{T}_j$ for all but one $j^* \in \mathcal{H}'$, and define

$$\hat{T}_{j^*} = T - T_{\mathcal{C}} - \sum_{j \in \mathcal{H}' \setminus \{j^*\}} \hat{T}_j.$$

The simulator then outputs fresh encryptions $\text{Enc}_{\text{pk}}(\hat{T}_j)$ for honest parties. Their sum encrypts exactly T .

Because individual honest local states are never decrypted, semantic security of the encryption scheme implies that replacing encryptions of the real honest T_j values by encryptions of another decomposition having the same final aggregate is computationally indistinguishable to the adversary. Rerandomization ensures that the ciphertext representations are fresh and cannot be linked to intermediate conditional-update states. The only ciphertext whose plaintext is eventually revealed is the aggregate $\tilde{T}$, and its plaintext is exactly the state T determined by U and X_1 .

If $\mathcal{H}' = \emptyset$, then every party P_j , $j > 1$, is corrupted. In this case all local contributions are already determined by corrupted inputs, and there is no honest local sketch that must be simulated. The aggregate state is consequently determined entirely by the corrupted inputs together with X_1 , so simulation is immediate.

HE Collaborative Decryption. The parties collaboratively decrypt only the aggregate ciphertext $\tilde{T}$. By the assumed security of the multi-key decryption procedure, the transcript of partial decryption can be simulated from the ciphertext,

the corrupted parties' secret-key shares, and the prescribed decryption result when the receiver is corrupted.

If $P_1 \notin \mathcal{A}$, no plaintext result is given to the adversary, and the simulator produces the corrupted parties' partial-decryption views using the underlying decryption simulator. If $P_1 \in \mathcal{A}$, the simulator already knows $T = \text{IIS.Insert}(U \setminus X_1, 0)$ and simulates a collaborative-decryption transcript resulting in this value.

Importantly, the protocol never decrypts any individual $\tilde{T}_j$. Therefore, the receiver cannot learn the partition of the union elements among the local sketches. It learns only their aggregate T .

MPC Aggregation. We next consider the MPC realization. After the OIIS calls, party P_i holds shares $\langle T_j \rangle_i$ of the local states and computes

$$\langle T \rangle_i = \sum_{j=2}^n \langle T_j \rangle_i.$$

The parties then invoke $\mathcal{F}_{\text{ZeroShare}}$ to obtain fresh sketch-shaped additive zero shares $\langle 0 \rangle_1, \dots, \langle 0 \rangle_n$ satisfying $\sum_{i=1}^n \langle 0 \rangle_i = 0$. Party P_i sends $M_i = \langle T \rangle_i + \langle 0 \rangle_i$ to P_1 .

The fresh zero sharing rerandomizes the additive representation of the aggregate state. Indeed,

$$\sum_{i=1}^n M_i = \sum_{i=1}^n \langle T \rangle_i + \sum_{i=1}^n \langle 0 \rangle_i = T.$$

Thus, $(M_1, \dots, M_n)$ is simply a fresh additive sharing of the aggregate plaintext state T .

Again, consider first the case $P_1 \notin \mathcal{A}$. The corrupted parties know their own aggregate shares, zero shares, and outgoing messages M_i , but they do not receive the messages of honest parties sent to P_1 . The simulator uses the assumed simulator for $\mathcal{F}_{\text{ZeroShare}}$ to generate the corrupted parties' zero shares and computes their outgoing messages exactly as in the real execution. Because at least one share required to reconstruct T remains unavailable to the coalition, these values reveal no aggregate plaintext state to the corrupted non-receiver parties.

Now suppose $P_1 \in \mathcal{A}$. The simulator receives U from the ideal functionality and therefore computes $Y = U \setminus X_1$ and the corresponding aggregate state $T = \text{IIS.Insert}(Y, 0)$. It must simulate all messages M_i received by the corrupted receiver.

For every corrupted party P_i , the simulator already has a simulated aggregate share $\langle T \rangle_i$ and zero share $\langle 0 \rangle_i$, so it sets $M_i = \langle T \rangle_i + \langle 0 \rangle_i$. For the honest parties, the messages can be sampled as a fresh additive sharing of the remaining portion of T . Let $\mathcal{H}$ denote the honest parties. Choose M_i uniformly for all but one honest party P_{i^*} and set

$$M_{i^*} = T - \sum_{i \in \mathcal{A}} M_i - \sum_{i \in \mathcal{H} \setminus \{i^*\}} M_i.$$

The resulting messages satisfy $\sum_{i=1}^n M_i = T$ and have exactly the distribution of a freshly rerandomized additive sharing of

T . Consequently, the corrupted receiver's view is distributed identically to its view in the real protocol.

This argument also covers the case in which only one party is honest. In that case, the honest message is uniquely determined by the requirement that all M_i sum to T . This does not reveal additional information because P_1 is already entitled to reconstruct T , and the honest message represents only the honest party's share of the aggregate state rather than any individual local state T_j .

The order of operations is important here. The parties first sum their shares across all local sketches and only then rerandomize the resulting aggregate share. Hence, the values sent to P_1 are shares of the global aggregate T , not shares corresponding separately to $T_2, \dots, T_n$. Therefore, even a corrupted receiver cannot use these messages to determine which party contributed a particular encoded element.

MPC Reconstruction. The receiver computes

$$T = \sum_{i=1}^n M_i.$$

If $P_1 \notin \mathcal{A}$, this value is unavailable to the adversary and requires no further simulation. If $P_1 \in \mathcal{A}$, the simulator has already constructed T from the prescribed ideal output U and X_1 , so the reconstructed state in the simulated execution is identical to the state expected by the corrupted receiver.

No local state T_j is reconstructed. The only reconstructed plaintext object is the aggregate T , which encodes $U \setminus X_1$.

Leakage of the Aggregate Sketch. It remains to argue that revealing the aggregate plaintext sketch T to P_1 does not reveal information beyond the ideal MPSU output. The aggregate state is obtained by applying the public IIS insertion algorithm to exactly the set

$$Y = U \setminus X_1.$$

The set Y is completely determined by the receiver's own input X_1 and its prescribed output U . Therefore, any deterministic aggregate state constructed from Y and public parameters can also be computed by the simulator. If the IIS construction uses public randomness, such as public hash functions or a public seed, the simulator samples or uses the same public randomness and obtains a state with the same distribution.

Hence, the plaintext sketch does not reveal the identity of the party that contributed any element. It records only a symmetric encoding of the elements in Y , all of which are already determined by the receiver's prescribed output.

Decoding and Final Output. The receiver applies the corresponding IIS decoder to T and obtains $Y = U \setminus X_1$. It then outputs $X_1 \cup Y = U$. Since Y is efficiently computable from X_1 and U , the decoding stage reveals nothing beyond the ideal output.

For a coalition containing P_1 , the simulator simply computes $Y = U \setminus X_1$ and simulates the deterministic decoding computation. For a coalition not containing P_1 , neither T , Y , nor U is revealed, so no additional simulation is required.

Composition. We have constructed simulators for each component of the protocol. The private prefix-membership executions are simulated using the security of $\mathcal{F}_{\text{mq-SSPPMT}}$; the conditional encoding executions are simulated using the security of Π_{OHS} ; the MPC rerandomization step is simulated using $\mathcal{F}_{\text{ZeroShare}}$; and the HE encryption and collaborative-decryption transcripts are simulated using the security of the underlying multi-key partially homomorphic encryption scheme. All remaining operations are local computations on values already available to the corresponding party.

By the sequential composition theorem, replacing these components one at a time by their ideal simulators changes the adversary's view only by a computationally indistinguishable amount. Therefore, there exists a PPT simulator $\mathcal{S}$ such that

$$\text{View}_{\mathcal{A}}^{\Pi_{\text{MPSU}}} \approx_c \text{Sim}_{\mathcal{A}}^{\mathcal{F}_{\text{MPSU}}},$$

where $\mathcal{F}_{\text{MPSU}}$ reveals $U = \bigcup_{j=1}^n X_j$ only to P_1 .

Thus, assuming the security of $\mathcal{F}_{\text{mq-SSPPMT}}$, $\mathcal{F}_{\text{ZeroShare}}$, the selected OHS realization, and the underlying HE or MPC primitives, Π_{MPSU} securely realizes the single-receiver MPSU functionality against any allowed semi-honest coalition.

D.4 Multi-Query SSPPMT

The MPSU protocol in Figure 11 relies on $\mathcal{F}_{\text{mq-SSPPMT}}$ to determine whether each query element has already appeared in the prefix union of the preceding parties' sets. We now describe a concrete realization of $\mathcal{F}_{\text{mq-SSPPMT}}$ based on the batched secret-shared private membership test $\mathcal{F}_{\text{bSSPMT}}$ together with secure MPC.

Let P_j be the query party and let $P_1, \dots, P_{j-1}$ be the preceding set holders. Party P_j holds a batch of queries $X_j = \{x_{j,1}, \dots, x_{j,m}\}$. For every $i < j$, parties P_i and P_j invoke Π_{bSSPMT} in parallel on X_i and X_j . For each query $x_{j,v}$, party P_i receives a binary share $b_{i,j,v}$ and party P_j receives $b_{j,i,v}$ such that $b_{i,j,v} \oplus b_{j,i,v} = \mathbf{1}[x_{j,v} \in X_i]$.

The pairwise membership results are then aggregated using a balanced binary OR tree. Each OR gate is evaluated securely using Beaver triples and the identity $a \vee b = a \oplus b \oplus (a \wedge b)$. At the end of this computation, the parties hold binary shares $\beta_{1,j,v}, \dots, \beta_{j,j,v}$ satisfying

$$\bigoplus_{i=1}^j \beta_{i,j,v} = \bigvee_{i=1}^{j-1} (b_{i,j,v} \oplus b_{j,i,v}) = \mathbf{1}\left[x_{j,v} \in \bigcup_{i=1}^{j-1} X_i\right].$$

Finally, the parties use daBits [15] to convert these binary shares into arithmetic shares $b_{1,v}, \dots, b_{j,v}$ such that $\sum_{i=1}^j b_{i,v} = \bigoplus_{i=1}^j \beta_{i,j,v}$. Hence, the arithmetic shares reconstruct to 1 exactly when $x_{j,v}$ belongs to the prefix union

$\bigcup_{i=1}^{j-1} X_i$, and to 0 otherwise. The complete protocol is shown in Figure 16. Security follows from the security of $\mathcal{F}_{\text{bSSPMT}}$, the Beaver-triple-based MPC evaluation of the OR tree, and the daBit-based share conversion. Hence, $\Pi_{\text{mq-SSPPMT}}$ securely realizes $\mathcal{F}_{\text{mq-SSPPMT}}$ against semi-honest adversaries with a dishonest majority.

Theorem 5 (Correctness and Security of $\Pi_{\text{mq-SSPPMT}}$). *Assume that Π_{bSSPMT} , the Beaver-triple-based MPC protocol, and the daBit-based share-conversion protocol are correct and secure against semi-honest adversaries with a dishonest majority. Then $\Pi_{\text{mq-SSPPMT}}$ correctly and securely realizes $\mathcal{F}_{\text{mq-SSPPMT}}$ against semi-honest adversaries with a dishonest majority.*

Proof. We first establish correctness. For every query $x_{j,v}$ and every preceding party P_i , $i < j$, correctness of Π_{bSSPMT} guarantees

$$b_{i,j,v} \oplus b_{j,i,v} = \mathbf{1}[x_{j,v} \in X_i].$$

Therefore, the balanced OR tree computes

$$\bigvee_{i=1}^{j-1} (b_{i,j,v} \oplus b_{j,i,v}) = \mathbf{1}\left[x_{j,v} \in \bigcup_{i=1}^{j-1} X_i\right].$$

Correctness of the daBit-based conversion then yields arithmetic shares $b_{1,v}, \dots, b_{j,v}$ satisfying

$$\sum_{i=1}^j b_{i,v} = \mathbf{1}\left[x_{j,v} \in \bigcup_{i=1}^{j-1} X_i\right].$$

Hence, the protocol correctly realizes $\mathcal{F}_{\text{mq-SSPPMT}}$.

For security, consider any semi-honest coalition containing at most $j-1$ parties. Each invocation of Π_{bSSPMT} reveals only binary shares of the corresponding pairwise membership result. The balanced OR tree is evaluated using secure Beaver-triple-based MPC, so all intermediate values remain secret-shared. The final daBit-based conversion securely transforms the binary sharing into an arithmetic sharing without revealing the underlying membership bit. By sequential composition of these secure subprotocols, the coalition's view can be simulated from its inputs and prescribed output shares. Therefore, $\Pi_{\text{mq-SSPPMT}}$ securely realizes $\mathcal{F}_{\text{mq-SSPPMT}}$ against semi-honest adversaries with a dishonest majority. $\square$

D.5 Security of Multi-Receiver UNITED

Theorem 6 (Security of Multi-Receiver UNITED). *Assume that the underlying single-receiver UNITED protocol is correct and secure against semi-honest adversaries with a dishonest majority, and that the multi-key decryption procedure, additive zero-sharing functionality $\mathcal{F}_{\text{ZeroShare}}$, and row-local decoding procedure are correct and secure. Then the distributed HE-based protocol $\Pi_{\text{Dist-Agg}}$ in Figure 18 and the*

distributed MPC-based protocol $\Pi_{\text{Dist-Decode}}$ in Figure 17 correctly and securely realize the multi-receiver MPSU functionality. In particular, every party outputs $U = \bigcup_{j=1}^n X_j$, while no allowed coalition learns information about the honest parties' inputs beyond its own inputs and the prescribed union output.

Proof. Let $U = \bigcup_{j=1}^n X_j$ denote the prescribed output. In the multi-receiver construction, party P_1 additionally contributes a protected sketch T_1 encoding X_1 , while for every $j > 1$, the local sketch T_j encodes $U_j^* = \{x_{j,q} \in X_j : b_{j,q} = 0\}$. By definition of the prefix-membership bits, $X_1, U_2^*, \dots, U_n^*$ are pairwise disjoint and satisfy $X_1 \cup \bigcup_{j=2}^n U_j^* = U$. Hence, by correctness and symmetry of the underlying IIS, the aggregate state $T = \sum_{j=1}^n T_j$ encodes exactly U .

Correctness. For the HE-based realization, party P_q receives $\tilde{T}_j|_{\mathcal{R}_q}$ from every P_j , $j \in [n]$, and computes $\tilde{T}|_{\mathcal{R}_q} = \sum_{j=1}^n \tilde{T}_j|_{\mathcal{R}_q}$. By correctness of homomorphic aggregation and collaborative decryption, $T|_{\mathcal{R}_q} = \sum_{j=1}^n T_j|_{\mathcal{R}_q}$. Since $\{\mathcal{R}_1, \dots, \mathcal{R}_w\}$ partitions all sketch rows, combining the reconstructed shards yields the complete state T , whose decoding returns exactly U .

For the MPC-based realization, each party P_i locally holds an additive share $\langle T \rangle_i$ of the aggregate state $T = \sum_{j=1}^n T_j$. The parties obtain fresh zero shares $\langle 0 \rangle_1, \dots, \langle 0 \rangle_n$ satisfying $\sum_{i=1}^n \langle 0 \rangle_i = 0$. For every $q \in [n]$, party P_i sends $M_{i,q} = (\langle T \rangle_i + \langle 0 \rangle_i)|_{\mathcal{R}_q}$ to P_q . Therefore, $\sum_{i=1}^n M_{i,q} = T|_{\mathcal{R}_q}$. Party P_q decodes its assigned rows to obtain Y_q . Since the row partition covers the complete sketch and T encodes U , $U = \bigcup_{q=1}^n Y_q$. Thus, every party obtains the prescribed union.

Security. Let $\mathcal{A} \subsetneq [n]$ be any semi-honest coalition of at most $n - 1$ parties. We construct a simulator $\mathcal{S}$ given the corrupted parties' inputs and the prescribed output U .

For the HE-based realization, the encrypted local sketches and intermediate aggregated ciphertexts are simulatable from the security of the multi-key encryption scheme and the underlying OIIS protocol. A corrupted shard owner P_q additionally learns the plaintext shard $T|_{\mathcal{R}_q}$. Since the aggregate state T encodes exactly U , this shard is determined by U , the public IIS parameters, the public hash functions, and the public row partition. The simulator can therefore construct the same aggregate state from U and reproduce every plaintext shard revealed to corrupted parties. The collaborative-decryption transcript is simulatable by security of the underlying multi-key cryptosystem.

For the MPC-based realization, the aggregate sketch remains additively secret-shared before reconstruction. Each transmitted message has the form $M_{i,q} = (\langle T \rangle_i + \langle 0 \rangle_i)|_{\mathcal{R}_q}$. The fresh zero sharing rerandomizes the individual additive shares without changing their sum. Hence, any collection of fewer than n rerandomized shares can be simulated subject only to the reconstructed aggregate shard. If a corrupted party

P_q reconstructs $T|_{\mathcal{R}_q}$ and decodes Y_q , both values are determined by U and the public parameters, and can therefore be reproduced directly by the simulator.

Finally, combining the decoded shards reveals only U , which is already the prescribed output of the multi-receiver functionality. Therefore, the distributed reconstruction procedures reveal no additional information about the honest parties' inputs.

By sequential composition with the secure underlying UNITED protocol, both distributed constructions securely realize the multi-receiver MPSU functionality against semi-honest adversaries with a dishonest majority. $\square$