Private and Oblivious Set and Multiset Operations*

Marina Blanton and Everaldo Aguiar
Department of Computer Science and Engineering
University of Notre Dame

Abstract

Privacy-preserving set operations and set intersection in particular are a popular research topic. Despite a large body of literature, the great majority of available solutions are available as two-party protocols and are not composable. In this work we design a comprehensive suit of secure multi-party protocols for set and multiset operations that are composable, do not assume any knowledge of the set by the parties carrying out the secure computation, and can be used for secure outsourcing. All of our protocols have complexity of $O(m \log m)$ for sets or multisets of size m, significantly improving their scalability in practice compared to other state of the art multi-party solutions of complexity $O(m^2)$. Our protocols are secure in the information theoretic sense and are designed to minimize the round complexity.

1 Introduction

The ability to securely perform set operations on private inputs has numerous applications. As an example, we mention computing intersection of databases belonging to different agencies or organizations, which by law or other provisions are not permitted to share their records in the clear, but want to compute the set of records common to both of them. This can be used in contexts ranging from finding passengers of an airline who appear in the national terrorist watch list to computing the list of customers common to two companies for more effective advertisement purposes. The importance of the topic is also evidenced by a significant body of prior work, see, e.g., [22, 36, 23, 20, 34, 29, 2] and others.

Work on privacy-preserving set operations started with the seminal work of Freedman at el. [22]. Consequently, many other publications appeared with the goal of extending the functionality or improving its performance. Secure protocols are known for set intersection (e.g., [22, 36, 28, 20, 34]), set union (e.g., [36, 23, 29]), set intersection cardinality or over threshold cardinality (e.g., [45, 18]), multiset element reduction ([36]), and others. The great majority of publications assume the two-party setting, in which Alice and Bob each possess a private set A and B, respectively, apply a set operation to A and B, and learn the result (or only one party learns the result). In such protocols, the knowledge of the private input A or B is essential for correctly recovering the result. While this problem formulation has a large number of applications, the existing solutions for set operations cannot be securely used as building blocks in larger protocols as they are not designed to be composable. Furthermore, when a set A or B is the result of prior secure computation and is not known to either party in the clear, the existing solutions do not apply either.

The literature that provides solutions for the multi-party setting [36, 23] likewise assumes that each participant has access to her private set in the clear. A notable exception is a few publications

*Portions of this work were sponsored by grant AFOSR-FA9550-09-1-0223 from the Air Force Office of Scientific Research.

that use the information-theoretic setting [38, 40, 39, 41], where the input set is distributed by each party to the set of participants. The recent emergence of cloud computing demands techniques for secure outsourcing that will allow the benefits of available cloud services to be utilized to the fullest extent, which otherwise might not be used due to the fear of information disclosure. In that setting the computational parties do not have access to the private inputs and it is essential that they do not learn any information about the data they process, while able to carry it out correctly. In other words, the computation needs to be data-independent or oblivious. From that point of view, it is desirable to have protocols that are both composable and can be used in outsourced tasks, which we set as one of our goals.

Our contributions. In this work, we provide secure multi-party protocols for a number of set and multiset operations, which are union, intersection, difference, and element reduction (for multisets). Besides computing the main functionality, we provide variants of the protocols that produce cardinality of the resulting (multi)set or compute over-the-threshold cardinality and produce a bit. Furthermore, our protocols can be used to always hide the size of input/output (multi)sets or the size can be revealed to make any computation that follows more efficient (since complexity of set operations is proportional to the size of their representation). Finally, we provide a generic conversion from a multiset to a set that allows us to run our protocols for secure set operations on multisets.

The advantages of our solutions over previously available results are as follows:

1. The requirement that each input set/multiset is known by a participant in the clear is removed. This implies that the elements of the input sets can be arbitrarily partitioned among the participants. The input sets can also be a result of prior privacy-preserving computation and are not known in the clear to any participant.

2. Our protocols are composable. Because both the inputs and outputs are split among the participants, our protocols can be composed an arbitrary number of times or they can be used as building blocks in larger computations.

3. No intermediate results or other information are revealed to the participants, which makes the solution suitable for secure computation outsourcing. In other words, the parties who provide the inputs and/or learn the results can be different from the parties carrying out secure computation. This is in contrast to prior results, where the knowledge of a set in the clear was essential for correctness of the computation.

4. Our solution provides natural support for hiding the sizes of the sets. The input sets can be padded for additional security, and the size of the result is never revealed, unless the parties decide to do otherwise.

5. Unlike most prior literature, our techniques make no use of expensive operations based on public-key cryptography and achieve information-theoretic security (assuming the existence of secure channels between the participants).

6. All of our protocols are efficient and have $O(m \log m)$ communication and computation complexity where m is the set size. Contrast this with significantly higher $O(m^2)$ overhead of all existing multi-party protocols, which limits their scalability.

Security of our protocols is shown in both passive (semi-honest or honest-but-curious) and active (malicious) adversarial models.

2 Related Work

Privacy-preserving set operations. The first custom protocols for securely computing the intersection of two datasets and the two-party cardinality set intersection were described by Freedman et al. [22] and are both based on the use of homomorphic encryption and polynomial representation of sets. Along with the novel approach, the authors also proposed an optimization using balanced hash functions that reduced the computation overhead to $O(m \ln \ln m)$, while the overall communication complexity of the protocols was $O(m)$. Here m represents the set size and $n \geq 2$ will denote the number of parties. Using the same polynomial representation and noticing the application of some of its mathematical properties, Kissner and Song [36] extended the previous work by building a framework of multiset operations which included set union, intersection and element reduction. The later was also the first to establish protocols that were secure against malicious players when three or more parties were involved, which was done via zero-knowledge proofs to ensure the participants did not deviate from the protocol. The protocols that were secure against honest-but-curious adversaries (and the set intersection protocol that was secure in the malicious model) presented in [36] had communication complexities of $O(n^2m)$. For a scenario containing $c < n$ dishonestly colluding players, the authors also described a set intersection protocol that is secure against honest-but-curious adversaries with communication complexity of $O(c^2m)$. With regards to the computation complexity, the protocols in [36] are $O(n^2m^2)$. An efficient set union protocol for the malicious adversary, with communication complexity of $O(n^2m^2 + n^3m)$ and $O(n)$ rounds, is given in [23]. Hong et al. [32] suggest yet another set union protocol for the malicious adversary, which is constant-round and uses a modified ElGamal cryptosystem that directly encrypts the polynomial elements and reduces the size of communication. Also with respect to malicious adversaries, [12, 29, 19] delineate protocols for privacy preserving set intersection in the two party setting. The approach in [19], an adaptation of the efficient intersection protocol presented in [20], was able to yield linear complexities (in the number of set elements) for both communication and computation. Although previous work ([33]) had achieved that same asymptotic complexity while not in the random oracle model, the authors of [19] support their claims that the underlying operations of that protocol masked in the big-O notation were much more expensive than those presented by them. One noticeable recent work [2] adds to all the has been presented the feature of hiding the size of the set held by the client (participant who learns the result) when computing intersections. In that scenario, the client incurs a small penalty that is proportional to the size of their set (m_c). The computation performed locally by that party will no longer be linear, but instead $O(m_c \log m_c)$.

Hazay and Lindell [28] proposed the first protocol for the two-party private set intersection that is based on oblivious pseudo-random functions (OPRFs). Such protocols functioned as follows: One of the participants (the server S) generates a PRF f_k based on a random key $k \in \mathbb{Z}$. The same player then produces the set $\mathcal{U} = \{u_1, \ldots, u_n\}$ where $u_i = f_k(s_i)$ and $s_i \in S$. A random permutation of $\mathcal{U}$ is sent to the client C which then engages in the OPRF computation of $f_k(c_i)$, where the elements $c_i \in C$, and at the end of which the server learns no extra information about C (except its size), and the client learns the result of $S \cap C = \{c_i \text{ s.t. } f_k(c_i) \in \mathcal{U}\}$. Though this protocol does not naturally extend to the multiparty setting, it allows the number of exponentiations required in the computation to be reduced, improving its efficiency. If we denote m_s and m_c to be the number of elements in the server's and client's sets respectively, and t to be the the size of the binary representation of input elements, the analysis show that both protocols in [28] are constant round and have communication and computation (modular exponentiations) complexities of $O(m_s \cdot t + m_c)$. The authors also point out that their protocol does not resort to random oracles when malicious adversaries are considered. An improvement to the previous protocol is presented by Jarecki and

Liu [33]. Their modified protocol, based on the Decisional q-Diffie-Hellman Inversion assumption, is secure in the Common Reference String (CRS) model and computation is done based on a secure RSA modulus. Furthermore, a similar protocol that replaces OPRFs with unpredictable functions is seen in [34].

In addition to the set operations described so far, a few other works propose protocols that compute the size of the (rather than the actual) intersection [36, 45, 39, 18, 43]. De Cristofaro et al. [18] suggest the most efficient one, where the two-party private set intersection cardinality is computed in the semi-honest adversary model with linear-time computation and communication.

There are also some publications that describe private set intersection protocols that work in the information-theoretic setting [38, 40, 39, 41]. Li and Wu [38] proposed the first protocol that was unconditionally secure. Their protocol assumed polynomial representation of sets, and the use of a two-dimensional secret sharing scheme (i.e., where values are shared among the players and each share is again shared among them). The communication complexity of the semi-honest adversary version of this protocol is $O(n^3m^2)$ (though its actual efficiency is contested in [40]). Patra et al. [40] present an information-theoretically secure private set intersection protocol with communication complexity of $O(n^3m^2 + n^4)$ and that works when $t < n/3$ parties are under control of an active adversary. This protocol is improved in [41] to work even when the adversary takes control of $t < n/2$ parties (communication complexity in that case becomes $O(n^4m^2 + n^5)$). Narayanan et al. [39] addresses private matching, set disjointness and cardinality of set intersection under information-theoretic settings with semi-honest adversaries corrupting $t < n/2$ parties. Lastly, we also highlight the work by Sang and Shen [43], which describes protocols for set intersection, cardinality of set intersection, element reduction, over-threshold set-union, and subset relation for the Universal Composability model with static adversaries and $O(n^2m^2)$ complexities. Table 1 provides a brief comparison between the most relevant protocols with regards to their complexities and functionality. Note that, while the existing multi-party protocols apply a set operation to n sets, our solutions are described for two data sets and therefore the complexities increase by a factor of $O(n)$ if n sets are to be used. All existing multi-party set operations, however, have complexities that are at least quadratic in the set size m, while we achieve notable $O(m \log m)$.

Table 1: Summary of most recent (and relevant) protocols for set operations.

Ref.	Operation	Computation	Communication	Multi-party	Public-key	Size hiding
[19]	PSI	$O(m)$	$O(m)$	No	Yes	No
[18]	PSI-CA	$O(m)$	$O(m)$	No	Yes	No
[2]	PSI	$O(m \log m)$	$O(m)$	No	Yes	Yes
[36]	PSI	$O(n^2m^2)$	$O(n^2m)$	Yes	Yes	No
[32]	PSU	$O(n^2m^2)$	$O(n^2m)$	Yes	Yes	Yes
[40]	PSI*	$O(n^3m^2 + n^4)$	$O(n^3m^2 + n^4)$	Yes	No	No

* Information Theoretically Secure

In the table, a solution is marked as size hiding if the sizes of the input sets can be protected by means of padding, which is the same as the protection mechanism used in this work. We note that work by Ateniese et al. [2] achieves a stronger notion of size hiding in which no information about one of the two input sets is revealed. We additionally achieve that information about the size of the output set (beyond the bounds imposed by the sizes of the (padded) input sets) is not revealed to the parties.

Secure multi-party computation. The literature on secure multi-party computation and function evaluation is very extensive and its review is beyond the scope of this work. In the multi-party setting, which is employed in this work, the available techniques are garbled circuit evaluation

(see, e.g., [25, 5]), computation based on linear secret sharing (see, e.g., [44, 10]), and threshold homomorphic encryption (see, e.g., [21, 17, 11]). In this work we employ techniques based on a linear sharing scheme and design efficient and information-theoretically secure protocols for set and multiset operations.

3 Preliminaries

3.1 Framework

In this work we use the multi-party setting in which $n > 2$ parties $P_1, \ldots, P_n$ jointly execute prescribed functionality on private inputs and outputs. We utilize a linear secret sharing scheme (such as Shamir secret sharing scheme [44]) for representation of, and secure computation over private values. To ensure composability of our protocols, we assume that prior to the computation, the parties P_1 through P_n hold their respective shares of the input and also compute shares of the output. Then any party holding a private input will produce shares of its values before the computation starts, and upon computation completion the computational parties P_1 through P_n send their shares to the entities that are entitled to learn the result. This gives flexibility to the problem setting in that the parties holding the inputs may be disjoint from the parties carrying out the computation (as in the case with outsourcing). Similarly, the parties receiving the output do not have to coincide with the input parties or computational parties.

Throughout this work we assume that parties $P_1, \ldots, P_n$ are connected by secure authenticated channels with each other. Each input and output party also establishes secure channels with P_1 through P_n. With a (n, t)-secret sharing scheme, any private value is secret-shared among n parties such that any $t + 1$ shares can be used to reconstruct it, while t or fewer parties cannot learn any information about the shared value, i.e., it is perfectly protected in information-theoretic sense. Therefore, the values of n and t should be chosen such that an adversary is unable to corrupt more than t computational parties.

In a linear secret sharing scheme that we utilize, a linear combination of secret-shared values can be performed by each computational party locally, without any interaction, but multiplication of two secret-shared values requires communication between all computational parties. In other words, if we let notation $[x]$ mean that value x is secret-shared among $P_1, \ldots, P_n$, operations $[x]+[y]$, $[x]+c$, and $c[x]$ are performed by each P_i locally on its shares of x and y, while computation of $[x][y]$ is interactive. The most common way of implementing a multiplication operation is by sending the total of $O(n^2)$ messages (where each participant sends $n-1$ messages, one to each other participant) using, for instance, the techniques of [24], but recent results [31, 4] lower the communication to $O(n)$ messages per multiplication at the cost of preprocessing. All operations are assumed to be performed in a field $\mathbb{Z}_p$ for a small prime p greater than the maximum value that needs to be used in the computation (i.e., the range of values for members of sets and multisets). Without loss of generality, we assume that the domain of set elements consists of integers greater than 0.

Performance of secure computation techniques is of grand significance, as protecting secrecy of data throughout the computation often incurs substantial computational costs. For that reason, besides security, efficient performance of the developed techniques is one of our prime goals. Normally, performance of a protocol in the current setting is measured in terms of two parameters: (i) the number of interactive operations (multiplications, distributing shares of a private value or opening a secret-shared value) necessary to perform the computation and (ii) the number of sequential interactions, i.e., rounds. We employ the same metrics throughout this work.

3.2 Building blocks

We now proceed with a brief description of building blocks which are used in our solutions, namely, oblivious sorting, comparisons, and prefix multiplication.

Oblivious sorting. When sorting is utilized in secure computation, the sequence of operations that the parties execute must be independent of the set they are sorting, or oblivious, to ensure that no information about the private data is revealed. While most sorting algorithms are not oblivious, sorting can be performed obliviously using a sorting network. Such techniques use compare-and-switch operations, which are fixed and independent of the input. In our setting, a compare-and-switch operation can be implemented as follows:

$$\begin{aligned}[s] &\leftarrow \mathsf{GE}([a],[b])\\ [c] &\leftarrow [s][b] + (1-[s])[a]\\ [d] &\leftarrow [s][a] + (1-[s])[b]\end{aligned}$$

where GE denotes a "greater than or equal" operation (detailed below). That is, after comparing two values a and b, their positions are switched if $a \geq b$ and left unchanged otherwise (c corresponds to $\min(a,b)$ and d corresponds to $\max(a,b)$).

Ajtai et al. [1] describe a sorting network with $O(m \log m)$ comparisons for a set of cardinality m, but it has a very high constant. More practically, Batcher's network [3] uses $O(m \log^2 m)$ comparisons and was the basis of secure multi-party sorting in [35]. More recent results [37, 26, 27] developed oblivious randomized sorting algorithms with $O(m \log m)$ comparisons and low constant which succeed with very high probability. Throughout the paper, we will assume that $O(m \log m)$ oblivious sorting of Goodrich [26] is used. We use notation $([y_1],\ldots,[y_m]) \leftarrow \mathsf{Sort}([x_1],\ldots,[x_m])$ to denote secure implementation of oblivious sorting in this framework.

Other protocols. In addition to oblivious sorting, we rely on other secure protocols from prior literature in this framework, which are as follows:

- $[b] \leftarrow \mathsf{Eq}([x],[y])$ is an equality protocol that on input two secret-shared values x and y outputs a bit b which is set to 1 iff $x = y$. The most efficient implementation of this operation that we are aware of is due to Catrina and de Hoogh [7] which uses $\ell + 4\log\ell$ interactive operations in 4 rounds to compare ℓ-bit integers, where most of the cost is input independent and can be performed ahead of time.

- $[b] \leftarrow \mathsf{GE}([x],[y])$ is a comparison protocol that on input two secret-shared ℓ-bit values x and y outputs a bit b which is set to 1 iff $x \geq y$. Efficient implementations of this function also exists, e.g., we can use the comparison protocol from [7] with 4 rounds and $4\ell - 2$ interactive operations, where precomputation can also reduce the cost.

- $([y_1],\ldots,[y_n]) \leftarrow \mathsf{PreMul}([x_1],\ldots,[x_n])$ computes prefix-multiplication, where on input a sequence of integers $x_1,\ldots,x_n$, the output consists of values $y_1,\ldots,y_n$, where each $y_i = \prod_{j=1}^{i} y_j$. Secure multi-party implementation of PreMul in [7] can be achieved using 2 rounds and $3\ell - 1$ interactive operations for ℓ-bit operands.

The complexities of Eq, GE, and PreMul functionalities cited above correspond to statistically secure protocols, but alternative implementations that achieve perfect secrecy are available as well. All other parts of our solutions are perfectly secure, and therefore by using perfectly secure implementations of these building blocks the overall solutions will be perfectly secure as well.

3.3 Security model

For each presented protocol, we define its secure functionality such that the parties carrying out the computation do not provide any input and do not receive any output. Instead, it is assumed that prior to the beginning of each protocol the parties with inputs will secret-share their sets among the parties carrying out the computation. Likewise, if the result of a computation is to be revealed to one or more parties, the parties performing the computation will send their shares to the output parties who reconstruct the result.

We next formally define security using the standard definition in secure multi-party computation for semi-honest adversaries. We will prove our techniques secure in the semi-honest model and will then show that standard techniques for making the computation robust to malicious behavior apply to all of our protocols.

Definition 1 *Let parties $P_1, \ldots, P_n$ engage in a protocol π that computes function $f(\mathsf{in}_1, \ldots, \mathsf{in}_n) = (\mathsf{out}_1, \ldots, \mathsf{out}_n)$, where in_i and out_i denote the input and output of party P_i, respectively. Let $\mathrm{VIEW}_\pi(P_i)$ denote the view of participant P_i during the execution of protocol π. More precisely, P_i's view is formed by its input and internal random coin tosses r_i, as well as messages $m_1, \ldots, m_t$ passed between the parties during protocol execution:*

$$\mathrm{VIEW}_\pi(P_i) = (\mathsf{in}_i, r_i, m_1, \ldots, m_t).$$

Let $I = \{P_{i_1}, P_{i_2}, \ldots, P_{i_t}\}$ denote a subset of the participants for $t < n$ and $\mathrm{VIEW}_\pi(I)$ denote the combined view of participants in I during the execution of protocol π (i.e., the union of the views of the participants in I). We say that protocol π is t-private in presence of semi-honest adversaries if for each coalition of size at most t there exists a probabilistic polynomial time simulator S_I such that

$$\{S_I(\mathsf{in}_I, f(\mathsf{in}_1, \ldots, \mathsf{in}_n)\} \equiv \{\mathrm{VIEW}_\pi(I), \mathsf{out}_I\},$$

where $\mathsf{in}_I = \bigcup_{P_i \in I}$, $\mathsf{out}_I = \bigcup_{P_i \in I}\{\mathsf{out}_i\}$, and $\equiv$ denotes computational indistinguishability.

4 Set Operations

This section presents our solutions for several set operations – namely, set intersection, union, and difference, as well as our multiset element reduction protocol. All other multiset operations are treated in consecutive sections.

Intuitively, computing an operation on two sets A and B without any knowledge about the values that these sets contain appears to be hard if fewer than m^2 comparisons are used (one comparison for each $a_i \in A$ and $b_j \in B$). Indeed, if any given pair of elements a_i, b_j have not been compared, then for arbitrary sets A and B the result is not guaranteed to be correct. If, however, the result is known to be correct with fewer than m^2 comparisons, then some information about the input sets must be known which violates our security requirements. We, however, notice that once data-oblivious sorting can be used as a building block, significantly more efficient solutions than $O(m^2)$ are possible. In particular, all of our set and multiset operations use $O(m \log m)$ interactive operations and their round complexity exceeds that of sorting by a small (additive) constant.

4.1 Core protocols

Set union. The first protocol that we describe computes the set union $C = A \cup B$, where $A = \{a_1, \ldots, a_{m_1}\}$, $B = \{b_1, \ldots, b_{m_2}\}$, and $C = \{c_1, \ldots, c_{m_1+m_2}\}$. Initially the elements of A and B are combined into a new set and subsequently sorted. Next, we need to eliminate duplicates, as we

wish to keep only a single instance of each item appearing in either of the sets. To accomplish this, our protocol looks at adjacent items in the sorted set, x_i and x_{i+1}. If the elements are the same, the first instance is erased by setting the corresponding item c_i in the resulting set to 0 (recall that 0 is not a valid element of A or B). The protocol makes no changes to those items that occur a single time.

Protocol 1. $[c_1], \ldots, [c_{m_1+m_2}] \leftarrow \mathsf{Union}([a_1], \ldots, [a_{m_1}], [b_1], \ldots, [b_{m_2}])$

1. $[x_1], \ldots, [x_{m_1+m_2}] \leftarrow \mathsf{Sort}([a_1], \ldots, [a_{m_1}], [b_1], \ldots, [b_{m_2}])$;
2. for $i = 1$ to $m_1 + m_2 - 1$ do in parallel
3. $\quad [u_i] \leftarrow \mathsf{Eq}([x_i], [x_{i+1}])$;
4. $\quad [c_i] \leftarrow [x_i](1 - [u_i])$;
5. $[c_{m_1+m_2}] \leftarrow [x_{m_1+m_2}]$;
6. return $[c_1], \ldots, [c_{m_1+m_2}]$;

Note that the computation in the Union protocol can be parallelized, and each element of the resulting set is computed independently of others. While this protocol provides the most basic version, we subsequently describe how the size of the set C can be reduced to contain only non-zero elements (the actual members of the intersection) if desired.

Set intersection. Following the set union logic, our protocol for set intersection works in a similar manner. This time, after sorting the combined set, we wish to erase (i.e., set to 0) each distinct element once (note that there will be either one or two instances of each distinct element). In our protocol we compare two consecutive elements x_i and x_{i+1} in the sorted set and keep x_i if they are equal. As before, computing all elements of the result $A \cap B$ proceeds in parallel, which is of grand importance because the size of A and B can be very large.

Protocol 2. $[c_1], \ldots, [c_{m_1+m_2}] \leftarrow \mathsf{Int}([a_1], \ldots, [a_{m_1}], [b_1], \ldots, [b_{m_2}])$

1. $[x_1], \ldots, [x_{m_1+m_2}] \leftarrow \mathsf{Sort}([a_1], \ldots, [a_{m_1}], [b_1], \ldots, [b_{m_2}])$;
2. for $i = 1$ to $m_1 + m_2 - 1$ do in parallel
3. $\quad [u_i] \leftarrow \mathsf{Eq}([x_i], [x_{i+1}])$;
4. $\quad [c_i] \leftarrow [u_i][x_i]$;
5. $[c_{m_1+m_2}] \leftarrow 0$;
6. return $[c_1], \ldots, [c_{m_1+m_2}]$;

Set difference. Our set difference protocol for computing $A \setminus B$ combines sets A and $A \cap B$ and sorts them. As a result, any value that appears twice in the resulting ordered multiset needs to be eliminated (both instances need to be erased). To achieve this, our protocol compares two adjacent items and sets both of them to 0 if the items are equal. This can be accomplished by executing $[u_i] \leftarrow \mathsf{Eq}([z_i], [z_{i+1}])$, where z_i's comprise the result of sorting concatenation of A and $A \cap B$, and then computing the elements c_i's of the set difference $A \setminus B$ as $[c_i] \leftarrow [z_i](1 - [u_i])$, $[c_{i+1}] \leftarrow [z_{i+1}](1 - [u_i])$. The above logic can be safely realized when the computation is executed sequentially, but it needs to be modified if we want to parallelize the computation. To achieve this, we make sure that the value of each c_i in the resulting set depends on the result of comparison of z_i with z_{i-1} and z_{i+1}, and each c_i is set only once. In particular, we set c_i to 0 if either u_{i-1} or u_i is true and it is set to z_i otherwise. Here we use computation $a \vee b = a + b - ab$ for Boolean OR.

Protocol 3. $[c_1], \ldots, [c_k] \leftarrow \mathsf{Diff}([a_1], \ldots, [a_{m_1}], [b_1], \ldots, [b_{m_2}])$

1. $[x_1], \ldots, [x_{m_1+m_2}] \leftarrow \mathsf{Int}([a_1], \ldots, [a_{m_1}], [b_1], \ldots, [b_{m_2}])$;
2. $[y_1], \ldots, [y_{2m_1+m_2}] \leftarrow \mathsf{Sort}([a_1], \ldots, [a_{m_1}], [x_1], \ldots, [x_{m_1+m_2}])$;
3. $[z_1], \ldots, [z_k] \leftarrow [y_{\max(m_1,m_2)+1}], \ldots, [y_{2m_1+m_2}]$;
4. for $i = 1$ to $k-1$ do in parallel $[u_i] \leftarrow \mathsf{Eq}([z_i], [z_{i+1}])$;
5. for $i = 2$ to k do in parallel $[v_i] \leftarrow [u_i][u_{i-1}]$;
6. $[c_1] \leftarrow [z_1](1 - [u_1])$;
7. for $i = 2$ to k do in parallel $[c_i] \leftarrow [z_i](1 - [u_i] - [u_{i-1}] + [v_i])$;
8. return $[c_1], \ldots, [c_k]$;

Element reduction. Element reduction is applied to a single multiset A, during which one instance of each distinct element is erased. The logic for its implementation is very similar to that of set intersection. The difference is that in the above set intersection protocol each distinct element appeared only once or twice in the combined set, while in this case, each element can appear any number of times. We therefore erase the elements in a different order. In particular, the first instance of each distinct element is erased. This is implemented by comparing two adjacent elements x_i and x_{i+1} in the sorted multiset and setting the element at position $i+1$ in the result, c_{i+1}, to 0 iff x_i and x_{i+1} differ (i.e., x_{i+1} is a new distinct element). For correctness, the first element c_1 is always set to 0. As before, computation of each element of the resulting multiset can proceed in parallel.

Protocol 4. $[c_1], \ldots, [c_m] \leftarrow \mathsf{Red}([a_1], \ldots, [a_m])$

---

1. $[x_1], \ldots, [x_m] \leftarrow \mathsf{Sort}([a_1], \ldots, [a_m])$;
2. $[c_1] \leftarrow 0$;
3. for $i = 1$ to $m-1$ do in parallel
4. $\quad [u_i] \leftarrow \mathsf{Eq}([x_i], [x_{i+1}])$;
5. $\quad [c_{i+1}] \leftarrow [u_i][x_{i+1}]$;
6. return $[c_1], \ldots, [c_m]$;

4.2 Protocol variants

The above protocols implement the basic functionality of multi-party set operations. In this section we show how they can be modified or extended to enable a number of new features.

Opening the result of a (multi)set operation. The output of the protocols presented in section 4.1 cannot be safely opened without leaking information about their inputs because the locations of erased items will be revealed. If the result is to be opened (e.g., when one of the above operations is the last operation in the computation), the parties will need to additionally sort the result, or randomly permute it, prior to opening to hide all patterns. To do so, the last line of each protocol in section 4.1 should be changed from

return $[c_1], \ldots, [c_k]$;

to

return $\mathsf{Sort}([c_1], \ldots, [c_k])$;

for the appropriate value of k.

Reducing the size of the result of a (multi)set operation. The way our protocols are specified does not reveal the size of the resulting set or multiset. In certain cases, however, for efficiency reasons it is desirable to reveal the size of the resulting set and eliminate all extra elements.

We distinguish between these two modes of computation by referring to them as length-hiding and length-preserving, respectively. To perform a length-preserving set or multiset operation, the parties need to follow each protocol as defined in section 4.1 after which the parties sort the outcome and discard 0 elements by comparing each of them to 0 and opening the result of the comparison. More precisely, each "return $[c_1], \ldots, [c_k]$" operation (for the appropriate value of k) in the original protocol now needs to be replaced with the following lines:

1. $[d_1], \ldots, [d_k] \leftarrow \mathsf{Sort}([c_1], \ldots, [c_k])$;
2. $S \leftarrow \emptyset$;
3. for $i = 1$ to k in parallel
4. $\quad [b] \leftarrow \mathsf{Eq}([d_i], 0)$;
5. $\quad b \leftarrow \mathsf{Open}([b])$;
6. $\quad$ if $(b = 0)$ $S \leftarrow S \cup [d_i]$;
7. return S;

The Open operation above corresponds to broadcasting the shares of its argument, so that all parties can reconstruct its value.

Computing (multi)set cardinality or over-the-threshold cardinality. Our basic protocols for set operations compute the resulting set, while in certain applications different information such as set cardinality needs to be computed. It is, however, rather straightforward to modify our protocols to instead compute the cardinality (i.e., $|A \cap B|$ for set intersection) or over-the-threshold cardinality (i.e., $|A \cap B| \stackrel{?}{>} T$ for set intersection and threshold T) of the resulting set. For completeness, we next describe such modifications, which give us even simpler protocols than the original versions.

To compute set union cardinality, it is no longer necessary to compute the c_i's in the Union protocol. Instead, it suffices to compute only the number of elements that differ from the next adjacent element in the combined sorted set $x_1, \ldots, x_{m_1+m_2}$. In particular, we replace lines 2–6 in Union with the following computation:

2. for $i = 1$ to $m_1 + m_2 - 1$ do in parallel
3. $\quad [u_i] \leftarrow \mathsf{Eq}([x_i], [x_{i+1}])$;
4. return $m_1 + m_2 - \sum_{i=1}^{m_1+m_2-1} [u_i]$;

The set union over-the-threshold cardinality can likewise be computing $\mathsf{GE}(T, m_1+m_2-\sum_{i=1}^{m_1+m_2-1}[u_i])$ and returning the complement of the obtained bit. The set intersection cardinality and reduced multiset cardinality protocols follow a similar logic, where now the parties need to compute and return $\sum_{i=1}^{m_1+m_2-1}[u_i]$ and $\sum_{i=1}^{m-1}[u_i]$, respectively. The over-the-threshold versions are formed analogously. Finally, to compute set difference cardinality, the parties need to produce the count of the number of elements that do not get erased from the resulting set. This can be achieved by replacing lines 6–8 of the Diff protocol with the following:

6. return $k - [u_k] + \sum_{i=2}^{k}([v_i] - 2[u_{i-1}])$;

As before, the over-the-threshold cardinality version is produced analogously.

4.3 Length-hiding set operations

Recall that our original protocols do not reveal any information about the size of the resulting set. To ensure that they can be used in the full length-hiding mode, we need to make sure that our protocols work correctly when the length of the input sets is also protected. To hide the actual length of a set, one adds to that set a number of additional elements that have value 0. In this

framework, for instance, all sets can be padded to be of the same size (or one of few fixed sizes). It remains to show that correctness of our protocols is preserved when the input sets contain dummy zero elements. We consider each protocol in turn.

In the Union protocol, after the step 1, all dummy elements will occupy the lowest indices in the sorted set which we denote 1 through s. During the loop execution, the zero elements will be set to zero again, which has no effect on the result of the operation. The only place where a care needs to be exercised is during a comparison of zero element x_s and the next non-zero element x_{s+1}. Notice that in the Union protocol, the result of computing $\mathsf{Eq}([x_s], [x_{s+1}])$ has no effect on x_{s+1}. We therefore obtain that the output of the protocol will be correct regardless of the number of regular elements contained in the sets A and B (including the case when A and B are entirely composed of dummy elements). By applying the same reasoning to other protocols, we obtain that regardless of whether zero elements are reset to zero or their values are preserved, the result of the computation is not affected. In the intersection protocol Int we have that computation "at the border" of dummy and regular elements, namely using x_s and x_{s+1}, has no effect on x_{s+1} and therefore the protocol works as expected on padded inputs. This is not the case in the element reduction protocol Red, but we can see that the result of $\mathsf{Eq}([x_s], [x_{s+1}])$ will be 0 and x_{s+1} will be set to 0 as required. Finally, in the set difference protocol Diff, the value of u_s will be 0 as well and therefore will not affect the correctness of the value of c_{s+1}.

We conclude that all of our protocols can be used unmodified on inputs padded with zero elements so that the size of both the input sets and the output set is protected.

4.4 Security

Correctness of the computation has been discussed with each respective protocol, and we now proceed with showing our protocols secure.

First, we note that the linear secret sharing scheme achieves perfect secrecy in presence of collusions of size at most t (i.e., zero information can be learned about secret-shared values by t or fewer parties) in the case of passive adversaries. Similarly, the multiplication product protocols does not reveal any information, as the only information transmitted to the participants are the shares. Furthermore, because other building blocks used in this work (namely, Eq, GE, PreMul, and Sort) have been previously shown to be secure, information is not revealed during their execution as well. We obtain that our protocols combine only secure building blocks without revealing any information to the computational parties (i.e., they only receive shares which information-theoretically protect private values). By Cannetti's composition theorem [6], we obtain that composition of secure sub-protocols results in security of the overall solution. More formally, to comply with the security definition in section 3.3, we can build a simulator for our protocols by invoking simulators for the corresponding building blocks to result in the environment that will be indistinguishable from the real protocol execution by the participants.

The only exception from the above is the extension to set operation protocols that allow the parties to learn information about the actual size of the resulting set, but this is intended by design and is considered to be acceptable.

To show security in presence of malicious adversaries, we need to ensure that (i) all participants prove that each step of their computation was performed correctly and that (ii) if some dishonest participants quit, others will be able to reconstruct their shares and proceed with the rest of the computation. The above is normally achieved using a verifiable secret sharing scheme (VSS), and a large number of results have been developed over the years (e.g., [24, 9, 30, 31, 4, 16, 14, 15] and many others). In particular, because any linear combination of shares is computed locally, each participant is required to prove that it performed each multiplication correctly on its shares. Such

results normally work for $t < \frac{n}{3}$ in the information theoretic or computational setting with different communication overhead and under a variety of assumptions about the communication channels. Additional proofs associated with robust multi-party computation include proofs that shares of a private value were distributed correctly among the participants (when the dealer is dishonest) and proofs of proper reconstruction of a value from its shares (when not already implied by other techniques). In addition, if at any point of the computation the participants are required to input values of a specific form, they would have to prove that the values they supplied are well formed. Such proofs are not necessary for the computation that we use to construct our protocols, but are needed by the implementations of some of the building blocks that we cite (such as comparison protocols).

Thus, security of our protocols in the malicious model can be achieved by using standard VSS techniques, e.g., [24, 10], where a range proof, e.g., [42] will be additionally needed for the building blocks. These VSS techniques would also work with a malicious input parties (who distribute input sets or multisets among the computational parties), who would need to prove that they generate legitimate shares of their data.

5 General Conversion from a Multiset to a Set

Our previous protocols do not meet correctness requirements when they are run on multisets. To enable computation on multisets, we describe a general conversion from a multiset to a set, which will allow all previous protocols to be run on multisets in this new representation with only notational changes.

Our solution converts a multiset $a_1, \ldots, a_m$ to a representation $\langle x_1, y_1\rangle, \ldots, \langle x_m, y_m\rangle$, where x_i's correspond to the a_i's, and indices y_i's count the number of instances of each distinct value in the multiset. That is, if a value v appears k times in the multiset, the indices of the corresponding elements in the multiset will be numbered 1 through k. This makes each pair $\langle x_i, y_i\rangle$ unique and our protocols for set operations apply.

Protocol 5. $\langle [x_1], [y_1]\rangle, \ldots, \langle [x_m], [y_m]\rangle \leftarrow \mathsf{M2S}([a_1], \ldots, [a_m])$

1. $[x_1], \ldots, [x_m] \leftarrow \mathsf{Sort}([a_1], \ldots, [a_m])$;
2. $[y_1] \leftarrow 1$;
3. for $i = 1$ to $m - 1$ do
4. $\quad [u_i] \leftarrow \mathsf{Eq}([x_i], [x_{i+1}])$;
5. $\quad [y_{i+1}] \leftarrow [u_i][y_i] + 1$;
6. return $\langle [x_1], [y_1]\rangle, \ldots, \langle [x_m], [y_m]\rangle$;

In this protocol, the indices y_i have to be computed sequentially. In the attempt to design an algorithm that does not require the number of rounds to be linear in the size of the multiset, we resort to the techniques that were used in [13] to design constant-round protocols for other integer arithmetic operations such as bit decomposition, addition of bit-decomposed integers, etc. In particular, suppose we are given an associative binary operator $\circ$. Also suppose that we can securely compute $\circ_{i=1}^{m}[a_i]$ in R rounds and $C(m)$ operations. Chandra et al. [8] describe a method for computing prefix-$\circ$, $\mathsf{Pre}_\circ$, in $2R$ rounds and $\sum_{i=1}^{\log_2 m} 2^i C(m \cdot 2^{-i}) + mC(\log_2 m) \leq \log_2 m C(m) + nC(\log_2 m)$ operations. Secure prefix-$\circ$ functionality is defined as $([b_1], \ldots, [b_m]) \leftarrow \mathsf{Pre}_\circ([a_1], \ldots, [a_m])$, where $b_i = \circ_{j=1}^{i} a_j$. In our context, this means that if we define a procedure for computing $\langle x_m, y_m\rangle$ in the multiset-to-set conversion using R rounds, we will be able to use the above method to compute all $\langle x_i, y_i\rangle$ in $2R$ rounds.

Before we proceed with further description, we need to specify the operator $\circ$ itself that we can use to perform the conversion. The procedure in the M2S protocol can be viewed as starting with individual elements, each with count 1, and aggregating the first i of them to compute the count at position i. Because the operator must be able to work on "individual" as well as "aggregate" values, we define it as follows:

$\langle [c_1], [c_2] \rangle \leftarrow \langle [a_1], [a_2] \rangle \circ \langle [b_1], [b_2] \rangle$

---

1. $[u] \leftarrow \mathsf{Eq}([a_1], [b_1])$;
2. $[c_1] \leftarrow [b_1]$;
3. $[c_2] \leftarrow [u][a_2] + [b_2]$;
4. return $\langle [c_1], [c_2] \rangle$;

The above assumes that the operands are well-formed, i.e., $b_1 \geq a_1$. We refer to this operation as addition with reset (i.e., the count is reset if the value of the current multiset element has changed, and the count is incremented otherwise). The operator can be shown to be associative.

To be able to use the method from [8] for computing $\mathsf{Pre}_\circ([a_1], \ldots, [a_m])$ using a solution to $\circ_{i=1}^m [a_i]$, we need a constant round procedure for computing $\circ_{i=1}^m [a_i]$, where $a_i = \langle x_i, y_i \rangle$. We realize it as show below. Note that in this protocol each y_i can be an arbitrary count (i.e., it $y_i > 1$, the pair $\langle x_i, y_i \rangle$ corresponds to an "aggregate" of several multiset elements with the same value), but the x_i's must form a non-decreasing sequence.

Protocol 6. $\langle [x], [y] \rangle \leftarrow \circ_{i=1}^m \langle [x_i], [y_i] \rangle$

---

1. for $i = 1$ to $m - 1$ do in parallel $[u_i] \leftarrow \mathsf{Eq}([x_i], [x_{i+1}])$;
2. $([v_{m-1}], \ldots, [v_1]) \leftarrow \mathsf{PreMul}([u_{m-1}], \ldots, [u_1])$;
3. for $i = 1$ to $m - 1$ do in parallel $[w_i] \leftarrow [v_i][y_i]$;
4. $[y] \leftarrow [y_m] + \sum_{i=1}^{m-1} [w_i]$;
5. $[x] \leftarrow [x_m]$;
6. return $\langle [x], [y] \rangle$;

In the protocol above, as a result of prefix multiplication in step 2, we obtain an array of bits $v_{m-1}, \ldots, v_1$, where v_i is set to 1 iff all elements x_i through x_m are equal. This allows us to count the number of elements in the input which have the same value as x_n. Their corresponding counts are added together in step 4 and are returned as the count for the entire set. This computation in particular implies that if $x_m > x_{m-1}$, then the pair $\langle x_m, y_m \rangle$ will be returned as required. This protocol allows us to obtain a new protocol for multiset-to-set conversion where the round complexity is the round complexity of sorting plus a small constant.

Protocol 7. $\langle [x_1], [y_1] \rangle, \ldots, \langle [x_m], [y_m] \rangle \leftarrow \mathsf{M2S}([a_1], \ldots, [a_m])$

---

1. $[x'_1], \ldots, [x'_m] \leftarrow \mathsf{Sort}([a_1], \ldots, [a_m])$;
2. for $i = 1$ to m do in parallel $[y'_i] \leftarrow 1$;
3. $\langle [x_1], [y_1] \rangle, \ldots, \langle [x_m], [y_m] \rangle \leftarrow \mathsf{Pre}_\circ(\langle [x'_1], [y'_1] \rangle, \ldots, \langle [x'_m], [y'_m] \rangle)$;
4. return $\langle [x_1], [y_1] \rangle, \ldots, \langle [x_m], [y_m] \rangle$;

This concludes our description of the conversion procedure. To illustrate how it can be used to perform multiset operations such as union, intersection, and difference, we sketch such protocols next. The first protocol that we demonstrate is for multiset union $A \cup B$. It assumes that the input

multisets are already available in the proper format with numbered instances of each distinct value. This can be achieved by executing the conversion protocol twice as $\langle [x'_1], [y'_1] \rangle, \ldots, \langle [x'_{m_1}], [y'_{m_1}] \rangle \leftarrow \mathsf{M2S}([a_1], \ldots, [a_{m_1}])$ and $\langle [x''_1], [y''_1] \rangle, \ldots, \langle [x''_{m_2}], [y''_{m_2}] \rangle \leftarrow \mathsf{M2S}([b_1], \ldots, [b_{m_2}])$. Alternatively, the input multisets might already be available in the proper format as a result of prior processing. For instance, the output of the multiset union protocol presented next produces an (unsorted) multiset with properly numbered elements. The only exception to that are zero elements which have been erased as the result of union computation. In particular, their counts are also set to zero to ensure that such elements do not affect correctness of our protocols during their composition.

In the multiset union protocol below, we utilize a slightly modified sorting procedure which we denote by SortT. It refers to sorting where instead of individual integers, each element is represented as a (3-)tuple. The comparisons are done using the last element of each tuple, but the entire tuples are swapped based on the outcome of a comparison.

Protocol 8. $\langle [x_1], [y_1] \rangle \ldots, \langle [x_{m_1+m_2}], [y_{m_1+m_2}] \rangle \leftarrow \mathsf{MUnion}(\langle [x'_1], [y'_1] \rangle, \ldots, \langle [x'_{m_1}], [y'_{m_1}] \rangle, \langle [x''_1], [y''_1] \rangle, \ldots, \langle [x''_{m_2}], [y''_{m_2}] \rangle)$

---

1. $k \leftarrow \max(m_1, m_2)$;
2. $\langle [\alpha_1], [\beta_1], [\gamma_1] \rangle, \ldots, \langle [\alpha_{m_1+m_2}], [\beta_{m_1+m_2}], [\gamma_{m_1+m_2}] \rangle \leftarrow \mathsf{SortT}(\langle [x'_1], [y'_1], k[x'_1] + [y'_1] \rangle, \ldots, \langle [x'_{m_1}], [y'_{m_1}], k[x'_{m_1}] + [y'_{m_1}] \rangle, \langle [x''_1], [y''_1], k[x''_1] + [y'' + 1] \rangle, \ldots, \langle [x''_{m_2}], [y''_{m_2}], k[x''_{m_2}] + [y''_{m_2}] \rangle)$;
3. for $i = 1$ to $m_1 + m_2 - 1$ do in parallel
4. $\quad [u_i] \leftarrow \mathsf{Eq}([\gamma_i], [\gamma_{i+1}])$;
5. $\quad [x_i] \leftarrow [\alpha_i](1 - [u_i])$;
6. $\quad [y_i] \leftarrow [\beta_i](1 - [u_i])$; //optional
7. $[x_{m_1+m_2}] \leftarrow [\alpha_{m_1+m_2}]$;
8. $[y_{m_1+m_2}] \leftarrow [\beta_{m_1+m_2}]$; //optional
9. return $\langle [x_1], [y_1] \rangle, \ldots, \langle [x_{m_1+m_2}], [y_{m_1+m_2}] \rangle$;

As we indicate above, lines 6 and 8 are optional. That is, if the counts for each value do need to be maintained, the protocol returns only $[x_i]$'s. Otherwise, the counts can be computed at low cost (i.e., significantly lower than executing the $\mathsf{M2S}$ protocol).

The multiset intersection protocol, MInt, is substantially similar to the MUnion protocol. To obtain MInt, we need to replace lines 5–8 in MUnion with

5. $[x_i] \leftarrow [u_i][\alpha_i]$;
6. $[y_i] \leftarrow [u_i][\beta_i]$;
7. $[x_{m_1+m_2}] \leftarrow 0$;
8. $[y_{m_1+m_2}] \leftarrow 0$;

The complexities of MUnion and MInt are exactly the same. It is also not very difficult to derive multiset difference protocol MDiff from its set version Diff, which we provide next.

Protocol 9. $\langle [x_1], [y_1] \rangle \ldots, \langle [x_{m_1+m_2}], [y_{m_1+m_2}] \rangle \leftarrow \mathsf{MDiff}(\langle [x'_1], [y'_1] \rangle, \ldots, \langle [x'_{m_1}], [y'_{m_1}] \rangle, \langle [x''_1], [y''_1] \rangle, \ldots, \langle [x''_{m_2}], [y''_{m_2}] \rangle)$

---

1. $\ell \leftarrow \max(m_1, m_2)$;
2. $\langle [\alpha_1], [\beta_1] \rangle, \ldots, \langle [\alpha_{n+m}], [\beta_{n+m}] \rangle \leftarrow \mathsf{MInt}(\langle [x'_1], [y'_1] \rangle, \ldots, \langle [x'_{m_1}], [y'_{m_1}] \rangle, \langle [x''_1], [y''_1] \rangle, \ldots, \langle [x''_{m_2}], [y''_{m_2}] \rangle)$;
3. $\langle [\alpha'_1], [\beta'_1], [\gamma'_1] \rangle, \ldots, \langle [\alpha'_{2m_1+m_2}], [\beta'_{2m_1+m_2}], [\gamma'_{2m_1+m_2}] \rangle \leftarrow \mathsf{SortT}(\langle [x'_1], [y'_1], \ell[x'_1] + [y'_1] \rangle, \ldots, \langle [x'_{m_1}], [y'_{m_1}], \ell[x'_{m_1}] + [y'_{m_1}] \rangle, \langle [\alpha_1], [\beta_1], \ell[\alpha_1] + [\beta_1] \rangle, \ldots, \langle [\alpha_{m_1+m_2}], [\beta_{m_1+m_2}], \ell[\alpha_{m_1+m_2}] + [\beta_{m_1+m_2}] \rangle)$;
4. $\langle [\alpha''_1], [\beta'_1], [\gamma''_1] \rangle, \ldots, \langle [\alpha''_k], [\beta''_k], [\gamma''_k] \rangle \leftarrow \langle [\alpha'_{\ell+1}], [\beta'_{\ell+1}], [\gamma'_{\ell+1}] \rangle, \ldots, \langle [\alpha'_{2m_1+m_2}], [\beta'_{2m_1+m_2}], [\gamma'_{2m_1+m_2}] \rangle$;
5. for $i = 1$ to $k - 1$ do in parallel $[u_i] \leftarrow \mathsf{Eq}([\gamma''_i], [\gamma''_{i+1}])$;

6. for $i = 2$ to k do in parallel $[v_i] \leftarrow [u_i][u_{i-1}]$;
7. $[x_1] \leftarrow [\alpha''_1](1 - [u_1])$;
8. $[y_1] \leftarrow [\beta''_1](1 - [u_1])$; //optional
9. for $i = 2$ to k do in parallel
10. $\quad [x_i] \leftarrow [\alpha''_i](1 - [u_i] - [u_{i-1}] + [v_i])$;
11. $\quad [y_i] \leftarrow [\beta''_i](1 - [u_i] - [u_{i-1}] + [v_i])$; //optional
12. return $\langle [x_1], [y_1] \rangle, \ldots, \langle [x_k], [y_k] \rangle$;

Security of these protocols can be shown analogously to the security of set operations.

6 Direct Operations on Multisets

The previous section described efficient algorithms for private multiset operations using a general multiset-to-set conversion. It is, however, well known that often direct implementations are more efficient than utilizing general procedures. This is true for secure multiset operations as well. In particular, by directly computing a multiset operation both communication and round complexity is reduced approximately by a factor of two because sorting is used only once instead of calling it once for the conversion procedure for each input multiset and once on the combined multiset for the set operation itself. Therefore in this section we describe protocols that provide direct implementation of multiset operations.

To be able to perform a multiset operation, we first sort the concatenation of two input multisets in such a way that all elements from the first set appear before the elements with an equal value from the second set. It is accomplished by setting indices associated with the elements of the first multiset A to 0 and indices associated with the elements of the second multiset B to 1. We then use values $2a_i + 0$ and $2b_j + 1$ to compare two elements during sorting, where a_i's and b_j's are elements of A and B, respectively. This will ensure that all elements with the same original value will be grouped together in the sorted multiset, but in a such a way that the elements from the first multiset appear before the elements with the same value from the second multiset. After the sorting, we assign to all elements with the same value counts according to their position in the sorted multiset, where the count indicates the difference in the number of elements from the first multiset and the second multiset observed at the current position. That is, if the first occurrence of a distinct value comes from the first set, its count is set to one (and otherwise it is set to -1). When another element with the same value is observed, its count is incremented, but once elements from the second multiset with the same value are observed, the count will be decremented after each occurrence. For instance, a sorted multiset $\langle 1, 0 \rangle, \langle 2, 0 \rangle, \langle 2, 0 \rangle, \langle 2, 1 \rangle, \langle 2, 1 \rangle, \langle 2, 1 \rangle, \langle 3, 1 \rangle$ will be converted to the following multiset with counts $\langle 1, 0, 1 \rangle, \langle 2, 0, 1 \rangle, \langle 2, 0, 2 \rangle, \langle 2, 1, 1 \rangle, \langle 2, 1, 0 \rangle, \langle 2, 1, -1 \rangle, \langle 3, 1, -1 \rangle$. The first element with a negative count corresponds to an element from the second multiset for which there is no matching element from the first multiset. Then depending on what operation needs to be performed, either elements with non-negative or elements with negative counts might need to be erased. For instance, to compute multiset union, we erase all element with non-negative counts from the second multiset (i.e., erase the duplicates); to compute multiset intersection, we erase all element of the first multiset and all elements of the second multiset with negative counts (i.e., those that do not have matching elements from the first multiset); and to compute multiset difference, we use $A \cap B$ as the first multiset and A as the second multiset and then erase all elements of the first multiset and the elements with non-negative counts from the second multiset (i.e., erase all elements of $A \cap B$ and matching elements from A). The (non-private) algorithm below for direct multiset union illustrates the logic for computing multiset union, where the sorting procedure *Sort*

takes tuples and performs comparisons using the last element of each tuple, but the entire tuples are swapped during compare-and-exchange operations.

Algorithm 1. $c_1, \ldots, c_{m_1+m_2} \leftarrow \mathsf{DMUnion}(a_1, \ldots, a_{m_1}, b_1, \ldots, b_{m_2})$

1. $\langle x_1, y_1, z_1\rangle, \ldots, \langle x_{m_1+m_2}, y_{m_1+m_2}, z_{m_1+m_2}\rangle \leftarrow Sort(\langle a_1, 0, 2a_1\rangle, \ldots, \langle a_{m_1}, 0, 2a_{m_1}\rangle, \langle b_1, 1, 2b_1 + 1\rangle, \ldots, \langle b_{m_2}, 1, 2b_{m_2} + 1\rangle)$;
2. $count_1 \leftarrow 1 - 2y_1$;
3. for $i = 2$ to $m_1 + m_2$ do
4. if $(x_{i-1} = x_i)$
5. if (y_i) $count_i \leftarrow count_{i-1} - 1$;
6. else $count_i \leftarrow count_{i-1} + 1$;
7. else $count_i \leftarrow 1 - 2y_i$;
8. $c_1 \leftarrow x_1$;
9. for $i = 2$ to $m_1 + m_2$ do in parallel
10. if $(y_i \wedge (count_i \geq 0))$ $c_i \leftarrow 0$;
11. else $c_i \leftarrow x_i$;
12. return $c_1, \ldots, c_{m_1+m_2}$;

The two for loops can be easily combined into one (i.e., x_i's can be reset to 0 inside the first loop). We separate them for clarity of presentation: the computation in the first loop will be common to all of our multiset operations, and the computation in the second loop is specific to multiset union. We also note that, for the purposes of the above sequential algorithm, it is not necessary to maintain negative counts. Instead, all elements from the second multiset that have no matching elements from the first multiset can have the same count (e.g., -1). If the same count is used, the comparison on line 9 can be replaced with an equality check, which would result in a slightly more efficient implementation. We, however, need to maintain the exact counts for the elements coming from both multisets for the purposes of a constant-round implementation of this functionality.

To compute the multiset intersection instead of the union, it is sufficient to replace lines 8–11 above with the following lines:

8. $c_1 \leftarrow 0$;
9. for $i = 2$ to $m_1 + m_2$ do in parallel
10. if $(\neg y_i)$ $c_i \leftarrow 0$;
11. else if $(count_i < 0)$ $c_i \leftarrow 0$;
12. else $c_i \leftarrow x_i$;

To compute the multiset difference, the sorting on line 1 of the above algorithm needs to be run on the result of $A \cap B$ and A (in that order, so that the elements of $A \cap B$ have indices set to 0 and the elements of A have indices set to 1). Then lines 8–11 of the direct multiset union need to be replaced with the following:

8. for $i = 1$ to $m_1 + m_2$ do in parallel
9. if $(\neg y_i)$ $c_i \leftarrow 0$;
10. else if $(count_i \geq 0)$ $c_i \leftarrow 0$;
11. else $c_i \leftarrow x_i$;

All of the algorithms for performing multiset operations that we described so far are sequential and involve a linear number of rounds. To be able to compute these multiset operations in constant round, all that is necessary is to design a mechanism for computing all counts $count_i$ in constant

round. Using the intuition developed in the previous section, we define a new operator for the purposes of computing counts, which can be securely implemented on two operands as follows:

$\langle [x], [y], [count] \rangle \leftarrow \langle [x_1], [y_1], [count_1] \rangle \diamond \langle [x_2], [y_2], [count_2] \rangle$

---

1. $[u] \leftarrow \mathsf{Eq}([x_1], [x_2])$;
2. $[x] \leftarrow [x_2]$;
3. $[y] \leftarrow [y_2]$;
4. $[count] \leftarrow [u][count_1] + [count_2]$;
5. return $\langle [x], [y], [count] \rangle$;

In the above, each x_1 and x_2 is a multiset element and y_1 and y_2 are bits. It is expected that the inputs are well-formed, which means that $x_1 \leq x_2$ and if $x_1 = x_2$, then $y_1 \leq y_2$. Then we obtain that if $x_1 = x_2$, then the counts are simply added. Otherwise, $count_1$ is ignored and $count_2$ is used in the result. This operator can also be shown to be associative.

The last piece that remains before we are ready to present our direct implementations of private multiset operations is to show how to compute unbounded fan-in $\diamond$ operator $\diamond_{i=1}^{m} \langle x_i, y_i, count_i \rangle$ in a constant number of rounds. This can be accomplished in a similar way to computing $\circ_{i=1}^{m} \langle x_i, y_i \rangle$ in section 5. In more detail, we have:

Protocol 10. $\langle [x], [y], [count] \rangle \leftarrow \diamond_{i=1}^{m} \langle [x_i], [y_i], [count_i] \rangle$

---

1. for $i = 1$ to $m - 1$ do in parallel $[u_i] \leftarrow \mathsf{Eq}([x_i], [x_{i+1}])$;
2. $([v_{m-1}], \ldots, [v_1]) \leftarrow \mathsf{PreMul}([u_{m-1}], \ldots, [u_1])$;
3. for $i = 1$ to $m - 1$ do in parallel $[w_i] \leftarrow [v_i][count_i]$;
4. $[x] \leftarrow [x_m]$;
5. $[y] \leftarrow [y_m]$;
6. $[count] \leftarrow [count_m] + \sum_{i=1}^{m-1} [w_i]$;
7. return $\langle [x], [y], [count] \rangle$;

Finally, the multiset union protocol becomes:

Protocol 11. $[c_1], \ldots, [c_{m_1+m_2}] \leftarrow \mathsf{DMUnion}([a_1], \ldots, [a_{m_1}], [b_1], \ldots, [b_{m_2}])$

---

1. $\langle [x'_1], [y'_1], [z'_1] \rangle, \ldots, \langle [x'_{m_1+m_2}], [y'_{m_1+m_2}], [z'_{m_1+m_2}] \rangle \leftarrow \mathsf{SortT}(\langle [a_1], 0, 2[a_1] \rangle, \ldots, \langle [a_{m_1}], 0, 2[a_{m_1}] \rangle, \langle [b_1], 1, 2[b_1] + 1 \rangle, \ldots, \langle [b_{m_2}], 1, 2[b_{m_2}] + 1 \rangle)$;
2. for $i = 1$ to $m_1 + m_2$ do in parallel $[count'_i] \leftarrow 1 - 2[y_i])$;
3. $\langle [x_1], [y_1], [count_1] \rangle, \ldots, \langle [x_{m_1+m_2}], [y_{m_1+m_2}], [count_{m_1+m_2}] \rangle \leftarrow \mathsf{Pre}_{\diamond}(\langle [x'_1], [y'_1], [count'_1] \rangle, \ldots, \langle [x'_{m_1+m_2}], [y'_{m_1+m_2}], [count'_{m_1+m_2}] \rangle)$;
4. for $i = 2$ to $m_1 + m_2$ do in parallel $[u_i] \leftarrow \mathsf{GE}([count_i], 0)$;
5. for $i = 2$ to $m_1 + m_2$ do in parallel $[v_i] \leftarrow [x_i][y_i]$;
6. $[c_1] \leftarrow [x_1]$;
7. for $i = 2$ to $m_1 + m_2$ do in parallel $[c_i] \leftarrow [x_i] - [u_i][v_i]$;
8. return $[c_1], \ldots, [c_{m_1+m+2}]$;

To obtain private multiset intersection, we only need to replace lines 6–7 above with

6. $[c_1] \leftarrow 0$;
7. for $i = 2$ to $m_1 + m_2$ do in parallel $[c_i] \leftarrow (1 - [u_i])[v_i]$;

To form a private multiset difference protocol, in addition to first executing multiset intersection of A and B and then applying the sorting procedure to the multisets $A \cap B$ and A, one needs to change the loops on lines 4 and 5 of the above DMUnion protocol to start from $i = 1$ and replace lines 6–7 with the following:

6. for $i = 1$ to $n + m$ do in parallel $[c_i] \leftarrow [u_i][v_i]$;

As before, security of these protocols can be shown in both passive and active models using the same argument as in section 4.4.

7 Conclusions

This work is the first to provide a comprehensive suite of protocols for multi-party set and multiset operations that are composable and can be used for secure outsourcing. All of our solutions are secure in the information-theoretic sense and achieve low communication cost of $O(m \log m)$ for data sets of size m (while all previous multiparty protocols have the cost of $O(m^2)$) and were designed to minimize round complexity.

References

[1] M. Ajtai, J. Komlós, and E. Szemerédi. An $O(n \log n)$ sorting network. In *ACM Symposium on Theory of Computing (STOC)*, pages 1–9, 1983.

[2] G. Ateniese, E. De Cristofaro, and G. Tsudik. (If) size matters: Size-hiding private set intersection. In *Public Key Cryptography (PKC)*, volume 6571 of *LNCS*, pages 156–173, 2011.

[3] K. Batcher. Sorting networks and their applications. In *AFIPS Spring Joint Computer Conference*, pages 307–314, 1968.

[4] Z. Beerliov-Trubniov and M. Hirt. Perfectly-secure MPC with linear communication complexity. In *Theory of Cryptography Conference (TCC)*, pages 213–230, 2008.

[5] A. Ben-David, N. Nisan, and B. Pinkas. FairplayMP: A system for secure multi-party computation. In *ACM Conference on Computer and Communications Security (CCS)*, pages 257–266, 2008.

[6] R. Canetti. Security and composition of multiparty cryptographic protocols. *Journal of Cryptology*, 13(1):143–202, 2000.

[7] O. Catrina and S. de Hoogh. Improved primitives for secure multiparty integer computation. In *Security and Cryptography for Networks (SCN)*, pages 182–199, 2010.

[8] A. Chandra, S. Fortune, and R. Lipton. Unbounded fan-in circuits and associative functions. In *Annual ACM Symposium on Theory of Computing*, pages 52–60, 1983.

[9] R. Cramer, I. Damgård, S. Dziembowski, M. Hirt, and T. Rabin. Efficient multiparty computations secure against an adaptive adversary. In *Advances in Cryptology – EUROCRYPT*, pages 311–326, 1999.

[10] R. Cramer, I. Damgård, and U. Maurer. General secure multi-party computation from any linear secret-sharing scheme. In *Advances in Cryptology – EUROCRYPT*, pages 316–334, 2000.

[11] R. Cramer, I. Damgård, and J. Nielsen. Multiparty computation from threshold homomorphic encryption. In *Advances in Cryptology – EUROCRYPT*, pages 280–300, 2001.

[12] D. Dachman-Soled, T. Malkin, M. Raykova, and M. Yung. Efficient robust private set intersection. In *Applied Cryptography and Network Security (ACNS)*, pages 125–142, 2009.

[13] I. Damgård, M. Fitzi, E. Kiltz, J. Nielsen, and T. Toft. Unconditionally secure constant-rounds multi-party computation for equality, comparison, bits and exponentiation. In *Theory of Cryptography Conference (TCC)*, volume 3876 of *LNCS*, pages 285–304, 2006.

[14] I. Damgård, M. Geisler, M. Krøigaard, and J. Nielsen. Asynchronous multiparty computation: Theory and implementation. In *Public Key Cryptography (PKC)*, pages 160–179, 2009.

[15] I. Damgård, Y. Ishai, and M. Krøigaard. Perfectly secure multiparty computation and the computational overhead of cryptography. In *Advances in Cryptology – EUROCRYPT*, pages 445–465, 2010.

[16] I. Damgård, Y. Ishai, M. Krøigaard, J. Nielsen, and A. Smith. Scalable multiparty computation with nearly optimal work and resilience. In *Advances in Cryptology – CRYPTO*, pages 241–261, 2008.

[17] I. Damgård and M. Jurik. A generalisation, a simplification and some applications of Paillier's probabilistic public-key system. In *Public Key Cryptography (PKC)*, pages 119–136, 2001.

[18] E. De Cristofaro, P. Gasti, and G. Tsudik. Fast and private computation of set intersection cardinality. Cryptology ePrint Archive: Report 2011/141, 2011.

[19] E. De Cristofaro, J. Kim, and G. Tsudik. Linear-complexity private set intersection protocols secure in malicious model. In *Advances in Cryptology – ASIACRYPT*, volume 6477 of *LNCS*, pages 213–231, 2010.

[20] E. De Cristofaro and G. Tsudik. Practical private set intersection protocols with linear complexity. In *Financial Cryptography and Data Security (FC)*, volume 6052 of *LNCS*, pages 143–159, 2010.

[21] P.-A. Fouque, G. Poupard, and J. Stern. Sharing decryption in the context of voting or lotteries. In *International Conference on Financial Cryptography (FC)*, volume 1962 of *LNCS*, pages 90–104, 2000.

[22] M. Freedman, K. Nissim, and B. Pinkas. Efficient private matching and set intersection. In *Advances in Cryptology – EUROCRYPT*, volume 3027 of *LNCS*, pages 1–19, 2004.

[23] Keith Frikken. Privacy-preserving set union. In *Applied Cryptography and Network Security (ACNS)*, volume 4521 of *LNCS*, pages 237–252, 2007.

[24] R. Gennaro, M. Rabin, and T. Rabin. Simplified VSS and fast-track multiparty computations with applications to threshold cryptography. In *ACM Symposium on Principles of Distributed Computing (PODC)*, pages 101–111, 1998.

[25] O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game. In *ACM Symposium on Theory of Computing (STOC)*, pages 218–229, 1987.

[26] M. Goodrich. Randomized Shellsort: A simple oblivious sorting algorithm. In *ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1262–1277, 2010.

[27] M. Goodrich. Spin-the-bottle sort and annealing sort: Oblivious sorting via round-robin random comparisons. In *Workshop on Analytic Algorithmics and Combinatorics (ANALCO)*, 2011.

[28] C. Hazay and Y. Lindell. Efficient protocols for set intersection and pattern matching with security against malicious and covert adversaries. In *Theory of Cryptography Conference (TCC)*, volume 4948 of *LNCS*, pages 155–175, 2008.

[29] C. Hazay and K. Nissim. Efficient set operations in the presence of malicious adversaries. In *Public Key Cryptography (PKC)*, volume 6056 of *LNCS*, pages 312–331, 2010.

[30] M. Hirt and U. Maurer. Robustness for free in unconditional multi-party computation. In *Advances in Cryptology – CRYPTO*, pages 101–118, 2001.

[31] M. Hirt and J. Nielsen. Robust multiparty computation with linear communication complexity. In *Advances in Cryptology – CRYPTO*, pages 463–482, 2006.

[32] J. Hong, J. W. Kim, J. Kim, K. Park, and J. H. Cheon. Constant-round privacy preserving multiset union. Cryptology ePrint Achive Report 2011/138, 2011. http://eprint.iacr.org/2011/138.

[33] S. Jarecki and X. Liu. Efficient oblivious pseudorandom function with applications to adaptive OT and secure computation of set intersection. In *Theory of Cryptography Conference (TCC)*, volume 5444 of *LNCS*, pages 577–594, 2009.

[34] S. Jarecki and X. Liu. Fast secure computation of set intersection. In *International Conference on Security and Cryptography for Networks (SCN)*, volume 6280 of *LNCS*, pages 418–435, 2010.

[35] K. Jónsson, G. Kreitz, and M. Uddin. Secure multi-party sorting and applications. Cryptology ePrint Archive: Report 2011/122, 2011.

[36] L. Kissner and D. Song. Privacy-preserving set operations. In *Advances in Cryptology – CRYPTO*, volume 3621 of *LNCS*, pages 241–257, 2005.

[37] T. Leighton and C. Plaxton. Hypercubic sorting networks. *SIAM Journal of Computing*, 27(1):1–47, 1998.

[38] R. Li and C Wu. An unconditionally secure protocol for multi-party set intersection. In *ACNS*, volume 4521 of *LNCS*, pages 226–236, 2007.

[39] G. Narayanan, T. Aishwarya, A. Agrawal, A. Patra, A. Choudhary, and C. Rangan. Multi party distributed private matching, set disjointness and cardinality of set intersection with information theoretic security. In *Cryptology and Network Security (CANS)*, pages 21–40, 2009.

[40] A. Patra, A. Choudhary, and C. Rangan. Information theoretically secure multi party set intersection re-visited. In *Selected Areas in Cryptography*, pages 71–91, 2009.

[41] A. Patra, A. Choudhary, and C. Rangan. Round efficient unconditionally secure MPC and multiparty set intersection with optimal resilience. In *Progress in Cryptology – INDOCRYPT*, pages 398–417, 2009.

[42] K. Peng and F. Bao. An efficient range proof scheme. In *IEEE International Conference on Information Privacy, Security, Risk and Trust (PASSAT)*, pages 826–833, 2010.

[43] Y. Sang and H. Shen. Efficient and secure protocols for privacy-preserving set operations. *ACM Transactions on Information and System Security*, 13(1):9:1–9:35, 2009.

[44] A. Shamir. How to share a secret. *Communications of the ACM*, 22(11):612–613, 1979.

[45] J. Vaidya and C. Clifton. Secure set intersection cardinality with applications to association rule mining. *Journal of Computer Security*, 13(4):593–622, 2005.