Private and Oblivious Set and Multiset Operations*

Marina Blanton and Everaldo Aguiar
Department of Computer Science and Engineering
University of Notre Dame, Notre Dame, IN, USA
{mblanton,eaguiar}@nd.edu

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 the available solutions are two-party protocols and are not composable. In this work we design a comprehensive suite of secure multi-party protocols for set and multiset operations that are composable, do not assume any knowledge of the sets by the parties carrying out the secure computation, and can be used for secure outsourcing. All of our protocols have communication and computation complexity of $O(m \log m)$ for sets or multisets of size m, which compares favorably with prior work. Furthermore, we are not aware of any results that realize composable operations. Our protocols are secure in the information theoretic sense and are designed to minimize the round complexity.

Categories and Subject Descriptors

K.6.5 [**Management of Computing and Information Systems**]: Security and Protection

General Terms

Algorithms, Security

Keywords

Private set and multiset operations, secure multi-party computation and outsourcing, secret sharing, oblivious sorting.

1. INTRODUCTION

The ability to securely perform set operations on private inputs has numerous applications. As an example, we mention computing the 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 useful in contexts ranging from finding passengers of an airline who appear in the national no-fly list to computing the list of customers common to two companies for more effective advertising. The importance of the topic is also evidenced by a large body of prior work such as [28, 43, 29, 26, 41, 35, 3] and others.

Work on privacy-preserving set operations started with the seminal work of Freedman at el. [28]. 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., [28, 43, 34, 26, 41]), set union (e.g., [43, 29, 35]), set intersection cardinality or over the threshold cardinality (e.g., [55, 24]), multiset element reduction ([43]), and others. The great majority of publications assume the two-party setting, in which Alice and Bob each possessing 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. That is, a set operation has to comprise the entire computation as neither the output can remain private from both parties nor the existing solutions apply when set A or B is the result of prior secure computation and is not known to either party in the clear.

The literature that provides solutions for the multi-party setting [43, 29] likewise assumes that each participant has access to her private set in the clear. Even the publications that use the information-theoretic setting [46, 48, 47, 49] require each participant to create a polynomial from its input set prior to distributing it to other 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 still being able to correctly carry out the required operations. 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

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

ASIACCS '12, May 2–4, 2012, Seoul, Korea.

our goals.

Our contributions. In this work, we provide secure multi-party protocols for 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 the 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 our protocols for secure set operations to be run 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 the computation. This is in contrast with prior results, where the knowledge of a set in the clear was essential for protocol correctness.

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 sum of the input sets' sizes. This compares favorably with the existing solutions (which we detail below).

Security of our protocols is shown in both passive (also known as 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 data sets and the two-party set intersection cardinality were described by Freedman et al. [28]. They are based on the use of homomorphic encryption, polynomial representation of sets and balanced hash functions that result in $O(m \ln \ln m)$ computation and $O(m)$ communication for two parties. Here m is the set size and in what follows $n \geq 2$ will denote the number of parties. Kissner and Song [43] extended that work by building a framework of multiset operations which included set union, intersection, set intersection cardinality, and subset relation. The protocols secure against honest-but-curious adversaries and the set intersection protocol secure in the malicious model have communication and computation complexities of $O(n^2 m)$ and $O(n^2 m^2)$, respectively. Until very recently, available multi-party set operation protocols had complexity quadratic in the set size, but a few recent publications improve the efficiency of such protocols. In particular, Cheon et al. [13] proposed the first set intersection protocol that achieves non-quadratic costs with respect to set sizes for both communication and computation. Subsequently, Dachman-Soled et al. [18] achieved linear in the number of parties broadcasts and operations by representing sets as multivariate polynomials and adopting a round table communication paradigm. Set union protocols for the malicious adversary are proposed in [29, 38]. The former achieves communication complexity of $O(n^2 m^2 + n^3 m)$ in $O(n)$ rounds while the latter aims at reducing the size of communication.

Also with respect to malicious adversaries, [40, 17, 35, 26, 25] delineate protocols for privacy preserving set intersection in the two-party setting. The approach in [25] was able to yield linear (in m) complexities for both communication and computation. One noticeable recent work [3] adds to intersection operations the feature of completely hiding the size of the set (including the upper bound) held by the client (i.e., the participant who learns the result). In that scenario, however, the client performs $O(m_c \log m_c)$ operations, where m_c is the client's set size. A series of other protocols [43, 55, 47, 24, 51] focus only on computing the cardinality of the intersection. Lastly, another line of work [34, 40, 41] builds two-party private set intersection protocols based on oblivious pseudo-random functions (OPRFs).

Other relevant literature deals with protocols in the information-theoretic setting [46, 48, 47, 49]. Li and Wu [46] proposed the first such set intersection protocol for semi-honest adversaries. It assumed polynomial representation of sets, used a secret sharing scheme to distribute values among the players, and achieved communication complexity of $O(n^3 m^2)$. Patra et al. [48, 49] present two information-theoretically secure private set intersection protocols, the second of which is an optimization of the previous one. The communication complexity of the protocol proposed in [49] is $O(n^4 m^2 + n^5)$ with a constant number of rounds and resilience against $t < n/2$ corrupted parties. Lastly, we highlight the work by Sang and Shen [51], which describes protocols for most set operations in the Universal Composability model with static adversaries and $O(n^2 m^2)$ complexities, and the recent private set intersection implementation by Huang et al. [39] that despite being based on garbled circuits, have comparable performance to that of custom two-party protocols.

Table 1 provides a brief comparison of the most relevant protocols with respect to their complexities and functionality. Notations PSI and PMI stand for "private set intersection" and "private multiset intersection," respectively. U

Ref.	Operation	Computation	Communication	Multi-party	Public-key	Size hiding	Composable
[25]	PSI	$O(m)$	$O(m)$		$\surd$		
[24]	PSI-CA	$O(m)$	$O(m)$		$\surd$		
[3]	PSI	$O(m \log m)$	$O(m)$		$\surd$	$\surd$	
[13]	PSI	$O(n^3 m)$	$O(n^3 m)$	$\surd$	$\surd$		
[18]	PSI	$O(nm^2)$	$O(nm + m \log^2 m)$	$\surd$	$\surd$		
[38]	PMU	$O(n^2 m^2)$	$O(n^2 m)$	$\surd$	$\surd$	$\surd$	
[48]	PSI	$O(n^3 m^2 + n^4)$	$O(n^3 m^2 + n^4)$	$\surd$			
This work	PSI, PSI-CA, PSU, PSU-CA, PSDiff, PSDiff-CA, PER, PER-CA, PMI, PMI-CA, PMU, PMU-CA, PMDiff, PMDiff-CA	$O(n^2 m \log m)$	$O(n^2 m \log m)$	$\surd$		$\surd$	$\surd$

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

stands for "union," ER stands for "element reduction," and CA means "cardinality." All complexities are listed for the malicious adversary. 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. [3] 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. The complexity of the results in Table 1 that rely on public-key cryptography is measured in public-key operations reported in them and the security parameters for communication are implicit. The remaining solutions that do not rely on public-key operations (namely, [48] and this work) achieve information-theoretic security. All computation and communication complexities reflect the combined work and communication of *all 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., [31, 6]), computation based on linear secret sharing (see, e.g., [52, 15]), and threshold homomorphic encryption (see, e.g., [27, 23, 16]). In this work we employ techniques based on a linear sharing scheme and design efficient and information-theoretically secure protocols for set and multi-set operations.

Parallel set operations. Computing set operations has also been examined in the realm of parallel computing. Early works such as [44, 53] described solutions for set operations that utilized specially designed array structures to efficiently compute these operations directly in hardware. More recent parallel techniques such as [9] involve a careful arrangement of the data into random balanced binary trees. While these techniques allow set operations to be performed efficiently, they were not designed to be secure, are not data-oblivious, and do not naturally lend themselves to secure multi-party protocols.

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 [52]) for representation of and secure computation on 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 pair-wise secure authenticated channels. 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 shares reveal no 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 secret sharing scheme that we utilize, any 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 of them. In other words, if we let $[x]$ denote 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 multiplication is by sending the total of $O(n^2)$ messages (where each P_i sends $n - 1$ messages, one to each other participant) using, for instance, the techniques of [30], but recent results [37, 5] lower the communication to $O(n)$ messages per multiplication at the cost of preprocessing. We assume complexity $O(n^2)$ in our analysis.

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 of set and multiset elements). Without loss of generality, we assume that the domain of (multi)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, a sorting network is. Such techniques use a fixed (input-independent) sequence of compare-and-switch operations. In our setting, a compare-and-switch operation can be implemented as follows:

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

where GE denotes a "greater than or equal" operation (detailed below) which produces a bit. After comparing two values a and b, c corresponds to $\min(a,b)$ and d corresponds to $\max(a,b)$.

Ajtai et al. [2] 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 [4] uses $O(m \log^2 m)$ comparisons and was the basis of secure multi-party sorting in [42]. More recent results [45, 32, 33] developed oblivious randomized sorting algorithms with $O(m \log m)$ comparisons and low constants which succeed with very high probability. Another recent solution is due to Zhang [57], in which oblivious sorting is achieved in constant round using $O(m^2)$ or $O(mR)$ communication and computation, where $[0, R]$ is the range of numbers to be sorted.

Throughout the paper (and in the complexity analysis in particular), we will assume that $O(m \log m)$ oblivious sorting of Goodrich [32] 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 in our framework that we are aware of is due to Catrina and de Hoogh [11] 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 exist, e.g., we can use the comparison protocol from [11] 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 [11] uses 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.

Finally, another recent work due to Toft [54] provides equality and comparison protocols of sublinear (in ℓ) complexity. In particular, the equality protocol in [54] uses $O(\kappa)$ interactive operations in a constant number of rounds, where κ is a correctness parameter, and a comparison is performed using $O(\log \ell(\kappa + \log\log \ell))$ interactive operations in $O(\log \ell)$ rounds or using $O(\sqrt{\ell}(\kappa + \log \ell))$ interactive operations in a constant number of rounds for the same κ. These protocols are, however, more suitable for secure multi-party computation based on homomorphic encryption and are applicable to our setting only when $t = 1$.

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 computational parties 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 $m_1, \ldots, m_k$ passed between the parties during protocol execution:*

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

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 $\mathsf{in}_I = \bigcup_{P_i \in I}\{\mathsf{in}_i\}$, $\mathsf{out}_I = \bigcup_{P_i \in I}\{\mathsf{out}_i\}$, 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 the consecutive section.

Intuitively, computing an operation on sets A and B without any knowledge of 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 (explicitly or implicitly) 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 comparisons, then some information about the input sets must be known which violates our security requirements. Fortunately for us, relationships between some pairs a_i, b_j can be determined implicitly, based on other explicit comparisons of elements of A and B and eliminates the need for explicit m^2 comparisons. We notice that once data-oblivious sorting is used as a building block, we can realize all of our set and multiset operations using $O(m \log m)$ interactive operations (comparisons) 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 $A = \{a_1, \ldots, a_{m_1}\}$, $B = \{b_1, \ldots, b_{m_2}\}$, $C = \{c_1, \ldots, c_m\}$, and $m = m_1 + m_2$. Initially the elements of A and B are combined into a new set and subsequently sorted. Next, we 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. $[c_1], \ldots, [c_m] \leftarrow \mathsf{Union}([a_1], \ldots, [a_{m_1}], [b_1], \ldots, [b_{m_2}])$

1. $[x_1], \ldots, [x_m] \leftarrow \mathsf{Sort}([a_1], \ldots, [a_{m_1}], [b_1], \ldots, [b_{m_2}])$;
2. for $i = 1$ to $m - 1$ 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. $[c_m] \leftarrow [x_m]$;
6. return $[c_1], \ldots, [c_m]$;

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 union) if desired.

Set intersection. Following the set union logic, we could implement our protocol for set intersection in a similar manner. This time, after sorting the combined set of size $m = m_1 + m_2$, 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 its simplest form, in the protocol we could compare two consecutive elements x_i and x_{i+1} in the sorted set and keep x_i if they are equal. Huang et al. [39], however, notice that the size of the output set can be reduced in half if instead we compare each even element of the sorted set to its adjacent elements. Then if either comparison results in 1, we keep the current element and otherwise set it to 0. The output consists of only even elements, which gives us $\lfloor m/2 \rfloor$ elements in the output set. Implementing this logic in our framework results in similar (in fact, slightly more efficient) performance compared to the simpler logic, but the output size is reduced in half, which improves efficiency of the computations that follow. We also note that from the set operations that we implement in this work, set intersection is the only operation where the output size can be reduced to a fraction of the input set sizes without any knowledge of the inputs by computing values at certain fixed locations.

In our set intersection protocol we implement the logic described above, where we have to make an exception for the last element in case $m = m_1 + m_2$ is even (i.e., in the case the element at position m is compared only to its predecessor at position $m - 1$). For any given element x_{2i} of the sorted set, let u_i denote the result of the comparison of x_{2i} with x_{2i-1} and v_2 denote the result of x_{2i}'s comparison with x_{2i+1}. Then to compute the corresponding element of the output set c_i, we need to multiply x_{2i} with the OR of u_i and v_i. In general, Boolean OR $a \vee b$ can be implemented as $a + b - ab$, but we note that in our case u_i and v_i will never be simultaneously 1. This means that the sum $u_i + v_i$ will correspond to their OR, reducing the number of interactive operations. 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. $[c_1], \ldots, [c_{\lfloor m/2 \rfloor}] \leftarrow \mathsf{Int}([a_1], \ldots, [a_{m_1}], [b_1], \ldots, [b_{m_2}])$

1. $[x_1], \ldots, [x_m] \leftarrow \mathsf{Sort}([a_1], \ldots, [a_{m_1}], [b_1], \ldots, [b_{m_2}])$;
2. for $i = 1$ to $\lfloor (m-1)/2 \rfloor$ do in parallel
3. $\quad [u_i] \leftarrow \mathsf{Eq}([x_{2i}], [x_{2i-1}])$;
4. $\quad [v_i] \leftarrow \mathsf{Eq}([x_{2i}], [x_{2i+1}])$;
5. $\quad [c_i] \leftarrow ([u_i] + [v_i])[x_i]$;
6. if $(m \bmod 2 = 0)$
7. $\quad [u_{m/2}] \leftarrow \mathsf{Eq}([x_m], [x_{m-1}])$;
8. $\quad [c_{m/2}] \leftarrow [u_{m/2}][x_m]$;
9. return $[c_1], \ldots, [c_{\lfloor m/2 \rfloor}]$;

Set difference. An intuitive solution to computing the set difference $A \setminus B$ is to combine sets A and $A \cap B$, sort the combined set, and eliminate all values that appear twice in the resulting multiset (by erasing both instances). This approach, however, results in running sorting twice (where sorting is executed on the set of size $2|A| + |B|$ the second time) and thus more than doubling the overhead compared to other protocols. Our solution instead is to label the elements of the two sets with opposite bits which will allow us to perform this asymmetric operation using a single sort. In detail, we associate a zero bit with each element of set

A and a bit with value 1 with each element of B. The concatenation of these $m = |A| + |B|$ tuples is then sorted using a slightly modified sorting procedure that we denote by SortT. In this case, the comparisons performed during the sorting process only take into consideration the first element of each (2-)tuple, but the entire tuples are swapped based on the outcome of a comparison.

After sorting, we compare (in parallel) each element of the sorted set to its successor and store the results into a bit vector u. Based on these results, the protocol will then erase (set to 0) each pair of elements that have the same value, while keeping those that have unique values unchanged. To erase both instances of duplicate elements, we can compute values c_i's by executing

$$[c_i] \leftarrow [x_i](1 - [u_i]); \quad [c_{i+1}] \leftarrow [x_{i+1}](1 - [u_i]);$$

for each i, where x_i's represent the previously sorted concatenation of the elements of A and B. Although this logic can be safely realized when the computation is executed sequentially, it needs to be modified if we want it to be parallelized. To achieve this, we make sure that the value of each c_i in the resulting set depends on the result of the comparison of x_i with x_{i-1} and x_{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 x_i otherwise. Similar to the OR computation in the set intersection, because at most one of u_{i-1} and u_i can be set for each value of i, the OR computation is performed as $u_{i-1} + u_i$ instead of full $u_{i-1} + u_i - u_{i-1}u_i$.

Finally, as the last step of the protocol we compute the elements c_i's of the set difference $A \setminus B$ by erasing all elements of B that still remain. This is accomplished using the second element of each tuple of the sorted set, which stores information about the input set from which the value originated.

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

1. $\langle [x_1], [y_1] \rangle, \ldots, \langle [x_m], [y_m] \rangle \leftarrow \mathsf{SortT}(\langle [a_1], [0] \rangle, \ldots, \langle [a_{m_1}], [0] \rangle, \langle [b_1], [1] \rangle, \ldots, \langle [b_{m_2}], [1] \rangle)$;
2. for $i = 1$ to $m - 1$ do in parallel $[u_i] \leftarrow \mathsf{Eq}([x_i], [x_{i+1}])$;
3. for $i = 2$ to $m - 1$ do in parallel $[c_i] \leftarrow [x_i](1 - [u_i] - [u_{i-1}])$;
4. $[c_1] \leftarrow [x_1](1 - [u_1])$;
5. $[c_m] \leftarrow [x_m](1 - [u_{m-1}])$;
6. for $i = 1$ to m do in parallel $[c_i] \leftarrow [c_i](1 - [y_i])$;
7. return $[c_1], \ldots, [c_m]$;

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 similar to that of the intuitive implementation of set intersection with the difference that in the 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 adjacent 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 result 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. Next, we show how they can be modified or extended to enable several new features.

Opening the result of a (multi)set operation. The output of the protocols 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 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. In the full version of this work [7] we also show how this can be performed more efficiently by using set compaction instead of full sorting.

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 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 specified after which they 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 $[c_1], \ldots, [c_k]$" statement (for appropriate k) in the original protocols needs to be replaced with:

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. As before, for efficiency reasons compaction can be used instead of sorting.

Computing (multi)set cardinality and over-the-threshold cardinality. Our basic protocols for set operations compute the resulting set, while in certain applications different information such as the 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., $|A \cap B| \stackrel{?}{\geq} T$

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 the 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 $x_1, \ldots, x_m$. In particular, we replace lines 2–6 in Union with the following computation:

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

The set union over-the-threshold cardinality can likewise compute and return $\mathsf{GE}(m - \sum_{i=1}^{m} [u_i], T)$.

The set intersection cardinality and the cardinality of a multiset after element reduction follow a similar logic, where now the parties need to compute and return $\sum_{i=1}^{\lfloor m/2 \rfloor} [u_i] + \sum_{i=1}^{\lfloor (m-1)/2 \rfloor} [v_i]$ and $\sum_{i=1}^{m-1} [u_i]$, respectively. The over-the-threshold versions are formed analogously to the corresponding set union version.

Finally, to compute the 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 3–7 of the Diff protocol with the following:

3. return $m_1 - \sum_{i=1}^{m-1} [u_i]$;

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

Performing set operations on multiple input sets. Our protocols for set union and intersection have been defined to work on two input sets, while existing literature on multi-party set operations considers the problem of computing set intersection or union of n input sets with n participating parties. Here we show that it is straightforward to modify our Union and Int protocols to work on k inputs for any $k \geq 2$ (i.e., k may or may not depend on n).

First, observe that a protocol for multiple-input set union $[c_1], \ldots, [c_m] \leftarrow \mathsf{Union}(a_1^{(1)}, \ldots, a_{m_1}^{(1)}, \ldots, a_1^{(k)}, \ldots, a_{m_k}^{(k)})$, where $C = \bigcup_{i=1}^{k} A^{(i)}$, $A^{(i)} = \{a_1^{(i)}, \ldots, a_{m_i}^{(i)}\}$ for $i = 1, \ldots, k$, and $m = \sum_{i=1}^{k} m_i$, can be obtained from the original **Protocol 1** with virtually no changes. The only obvious difference is that the first step of the protocol now consists of sorting the concatenations of all of the $A^{(i)}$'s, i.e., $[x_1], \ldots, [x_m] \leftarrow \mathsf{Sort}(a_1^{(1)}, \ldots, a_{m_1}^{(1)}, \ldots, a_1^{(k)}, \ldots, a_{m_k}^{(k)})$. As before, the algorithm keeps a single instance of each present distinct value and eliminates the rest.

To be able to implement a multiple-input set intersection, $[c_1], \ldots, [c_m] \leftarrow \mathsf{Int}(a_1^{(1)}, \ldots, a_{m_1}^{(1)}, \ldots, a_1^{(k)}, \ldots, a_{m_k}^{(k)})$, where now $C = \bigcap_{i=1}^{k} A^{(i)}$, the algorithm in **Protocol 2** needs to be modified. Because of the multiple input sets, we would like to keep only the elements that appear exactly k times in the sorted array. To accomplish that, instead of checking two consecutive elements, we need to compare two elements $k - 1$ positions apart. Similar to **Protocol 2**, instead of producing a set of size m, this time we output a set of size $\lceil (m-1)/k \rceil$ and the OR of multiple bits from which at most one is set is computed as their sum. More precisely, we obtain:

Protocol 5. $[c_1], \ldots, [c_{\lceil (m-1)/k \rceil}] \leftarrow \mathsf{Int}([a_1^{(1)}], \ldots, [a_{m_1}^{(1)}], \ldots, [a_1^{(k)}], \ldots, [a_{m_k}^{(k)}])$

1. $[x_1], \ldots, [x_m] \leftarrow \mathsf{Sort}([a_1^{(1)}], \ldots, [a_{m_1}^{(1)}], \ldots, [a_1^{(k)}], \ldots, [a_{m_k}^{(k)}])$;
2. for $i = 1$ to $m-k+1$ do in parallel $[u_i] \leftarrow \mathsf{Eq}([x_i], [x_{i+k-1}])$;
3. $d \leftarrow \lfloor (m-1)/k \rfloor$;
4. for $i = 1$ to d do in parallel $[c_i] \leftarrow \sum_{j=1}^{k} ([u_{(i-1)k+j}] \cdot [x_{(i-1)k+j}])$;
5. if $((m-1) \bmod k \neq 0)$ $c_{\lceil (m-1)/k \rceil} \leftarrow \sum_{j=1}^{(m-1) \bmod k} [u_{d \cdot k+j}][x_{d \cdot k+j}]$;
6. return $[c_1], \ldots, [c_{\lceil (m-1)/k \rceil}]$;

4.3 Length-hiding set operations

Recall that our original protocols do not reveal any information about the size of the resulting set (beyond the bounds implied by the sizes of the input sets). 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 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 Diff protocol, u_s will be 0 as well and therefore will not affect the correctness 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' security.

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 protocol does not reveal any information, as the only information transmitted to the partici-

pants are the shares. Furthermore, because other building blocks used in this work (i.e., 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 [10], we obtain that composition of secure sub-protocols results in security of the overall solution. More formally, to comply with the security definition, 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., [30, 14, 36, 37, 5, 22, 20, 21] 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 this setting 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., [30, 15], where a range proof, e.g., [50] will be additionally needed for the building blocks. These VSS techniques would also work with 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.

4.5 Performance

With the standard techniques for implementing multiplication (which the most basic interactive building block) in our setting such as [30], each multiplication has the combined cost of $O(n^2)$, which results in $O(n^2 m \log m)$ complexity of our (multi)set operation protocols. When the computational parties can be malicious, they will need to prove correctness of each multiplication, secret sharing, and possibly input reconstruction execution using VSS techniques, which also involves $O(n^2)$ communication and with recent techniques [37, 5] is even $O(n)$. Such results are known for both computational and unconditionally secure settings. There are also recent publications [22, 21] that achieve overhead only polylogarithmic in the number of parties n, but the complexity also includes a factor logarithmic in the overall amount of computation (i.e., the arithmetic circuit size).

To estimate the performance of our protocols, we can count the total number of interactive operations (rather than their asymptotic complexities). The number of comparisons used in Goodrich's and Batcher's oblivious sorting algorithms are $5m \log m$ and $\frac{1}{4} m \log^2 m$, respectively, for a set of size m [56]. Then each compare-and-switch operation can be implemented using $4\ell + 2$ interactive operations. We thus obtain that for $m = 1,000$, the protocols are dominated by $\approx 25,000$ compare-and-switch operations used in sorting, while for $m = 1,000,000$ we need to use $\approx 100,000,000$ such operations. Experimental results of optimized Sharemind tool [1] (which implements operations using slightly different arithmetic) with $n = 3$ and $\ell = 32$ on a LAN show that more a million of comparisons can be processed on the order of several seconds. This gives our solution good scalability even for input sets of significant size. Also, because performance of a protocol depends on the number of sequential interactive operations or rounds, we need to compute their number as well. With a $O(\log^2 m)$ depth of a sorting algorithm and 4 rounds for a compare-and-switch operation (using the comparison algorithm from [11]), the number of rounds is in low hundreds for sets of size 2^{10}–2^{20}. From the experiments in [8], one round of computation in this setting on a LAN takes on the order of 3–5 ms (for $n = 5$), which means that all rounds for sets of large size can be processed in the matter of several seconds or tens of seconds.

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. We also develop protocols for direct implementation of multiset operations without the need for converting them into a special form. These direct constructions are about twice as fast as first applying a general conversion procedure to the two input sets followed by a set operation. They are, however, omitted due to space considerations and can be found in the full version [7].

Our general 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 multiset elements 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 6. $\langle [x_1], [y_1] \rangle, \ldots, \langle [x_m], [y_m] \rangle \leftarrow \mathsf{M2S}([a_1], \ldots, [a_m])$

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 the above protocol, indices y_i's 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 [19] to design constant-round protocols for integer arithmetic operations. 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. [12] 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 $b_1 \geq a_1$. We call this operation addition with reset (i.e., the count is reset if the value of the current element has changed, and the counts are added otherwise). The operator can be shown to be associative.

To be able to use the method from [12] for computing $\mathsf{Pre}_\circ([a_1], \ldots, [a_m])$ from 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 shown below. Note that in this protocol each y_i can be an arbitrary count (i.e., if $y_i > 1$, $\langle x_i, y_i \rangle$ corresponds to an "aggregate" of several multiset elements with the same value), but the x_i's must form a non-decreasing sequence.

Protocol 7. $\langle [x], [y] \rangle \leftarrow \circ_{i=1}^{m} \langle [x_i], [y_i] \rangle$

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_m. 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 8. $\langle [x_1], [y_1] \rangle, \ldots, \langle [x_m], [y_m] \rangle \leftarrow \mathsf{M2S}([a_1], \ldots, [a_m])$

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 briefly describe 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 below produces a (not fully sorted) multiset with properly numbered elements. The only exception to that are zero elements in the result, the counts of which are also set to zero to ensure that they do not affect correctness of our protocols during their composition.

In the multiset union protocol below, we utilize the same SortT procedure as in **Protocol 3** with the difference that now longer tuples are sorted. As before, comparisons are done using the first element of each tuple, and the entire tuples are swapped based on the outcome of a comparison.

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

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

In the protocol k should be set to a value larger than any y'_i and y''_i (which are bounded by the size of the multisets). In that way, the values will be sorted by the first elements x'_i's and x''_i's, but in case of their equality, the ties will be resolved – and the tuples will be sorted – by the second elements y'_i's and y''_i's. The safest way to set k is therefore to use $k = \max(m_1, m_2) + 1$.

As we indicate above, lines 6 and 8 are optional. That is, if the counts for each value do not 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 M2S protocol).

The multiset intersection protocol, MInt, can be derived from **Protocol 2** using a similar approach. We obtain the following protocol, where m is used for $m_1 + m_2$:

Protocol 10. $\langle [x_1], [y_1], \rangle \ldots, \langle [x_{\lfloor m/2 \rfloor}], [y_{\lfloor m/2 \rfloor}] \rangle \leftarrow$ MInt $(\langle [x'_1], [y'_1] \rangle, \ldots, \langle [x'_{m_1}], [y'_{m_1}] \rangle, \langle [x''_1], [y''_1] \rangle, \ldots, \langle [x''_{m_2}], [y''_{m_2}] \rangle)$

1. $k \leftarrow \max(m_1, m_2) + 1$;
2. $\langle [\alpha_1], [\beta_1], [\gamma_1] \rangle, \ldots, \langle [\alpha_{m_1+m_2}], [\beta_{m_1+m_2}], [\gamma_{m_1+m_2}] \rangle \leftarrow$ SortT$(\langle k[x'_1]+[y'_1], [x'_1], [y'_1] \rangle, \ldots, \langle k[x'_{m_1}]+[y'_{m_1}], [x'_{m_1}], [y'_{m_1}] \rangle, \langle k[x''_1]+[y''_1], [x''_1], [y''_1] \rangle, \ldots, \langle k[x''_{m_2}]+[y''_{m_2}], [x''_{m_2}], [y''_{m_2}] \rangle)$;
3. for $i = 1$ to $\lfloor (m-1)/2 \rfloor$ do in parallel
4. $\quad [u_i] \leftarrow$ Eq$([\alpha_{2i}], [\alpha_{2i-1}])$;
5. $\quad [v_i] \leftarrow$ Eq$([\alpha_{2i}], [\alpha_{2i+1}])$;
6. $\quad [x_i] \leftarrow ([u_i] + [v_i])[\beta_i]$;
7. $\quad [y_i] \leftarrow ([u_i] + [v_i])[\gamma_i]$; //optional
8. if $(m \bmod 2 = 0)$
9. $\quad [u_{m/2}] \leftarrow$ Eq$([\alpha_m], [\alpha_{m-1}])$;
10. $\quad [x_{m/2}] \leftarrow [u_{m/2}][\beta_m]$;
11. $\quad [y_{m/2}] \leftarrow [u_{m/2}][\gamma_m]$; //optional
12. return $[x_1], \ldots, [x_{\lfloor m/2 \rfloor}]$;

It is also not very difficult to derive the multiset difference protocol MDiff from its set version Diff, which we provide next.

Protocol 11. $\langle [x_1], [y_1] \rangle \ldots, \langle [x_{m_1+m_2}], [y_{m_1+m_2}] \rangle \leftarrow$ 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. $\langle [\alpha_1], [\beta_1], [\gamma_1], [\delta_1] \rangle, \ldots, \langle [\alpha_{m_1+m_2}], [\beta_{m_1+m_2}], [\gamma_{m_1+m_2}], [\delta_{m_1+m_2}] \rangle \leftarrow$ SortT$(\langle \ell[x'_1]+[y'_1], [x'_1], [y'_1], [0] \rangle, \ldots, \langle \ell[x'_{m_1}]+[y'_{m_1}], [x'_{m_1}], [y'_{m_1}], [0] \rangle, \ldots, \langle \ell[x''_1]+[y''_1], [x''_1], [y''_1], [1] \rangle, \ldots, \langle \ell[x''_{m_2}] + [y''_{m_2}], [x''_{m_2}], [y''_{m_2}], [1] \rangle)$;
3. for $i = 1$ to m_1+m_2-1 do in parallel $[u_i] \leftarrow$ Eq$([\alpha_i], [\alpha_{i+1}])$;
4. $[x_1] \leftarrow [\beta_1](1 - [u_1])$;
5. $[y_1] \leftarrow [\gamma_1](1 - [u_1])$; //optional
6. for $i = 2$ to $m_1 + m_2$ do in parallel
7. $\quad [x_i] \leftarrow [\beta_i](1 - [u_i] - [u_{i-1}])$;
8. $\quad [y_i] \leftarrow [\gamma_i](1 - [u_i] - [u_{i-1}])$; //optional
9. for $i = 1$ to $m_1 + m_2$ do in parallel
10. $\quad [x_i] \leftarrow [\beta_i](1 - [\delta_i])$;
11. $\quad [y_i] \leftarrow [\gamma_i](1 - [\delta_i])$; //optional
12. return $\langle [x_1], [y_1] \rangle, \ldots, \langle [x_{m_1+m_2}], [y_{m_1+m_2}] \rangle$;

As before, we will only execute the lines marked as optional if the counts need to be preserved.

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

6. 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. They have natural support for hiding the size of a set operation's result and can be easily extended to compute cardinality or over-the-threshold cardinality of the result. All of our solutions are secure in the information-theoretic sense against malicious adversaries, achieve low communication and computation cost of $O(m \log m)$ for data sets of size m, and were designed to minimize round complexity.

7. REFERENCES

[1] Sharemind. http://sharemind.cyber.ee/.

[2] 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.

[3] 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.

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

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

[6] 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.

[7] M. Blanton and E. Aguiar. Private and oblivious set and multiset operations. IACR ePrint Archive Report 2011/464, 2011.

[8] M. Blanton and M. Aliasgari. Secure outsourced computation of iris matching. *Journal of Computer Security*, 2012.

[9] G. Blelloch and M. Reid-Miller. Fast set operations using treaps. In *ACM Symposium on Parallel Algorithms and Architectures*, pages 16–26, 1998.

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

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

[12] 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.

[13] J. H. Cheon, S. Jarecki, and J. H. Seo. Multi-party privacy-preserving set intersection with quasi-linear complexity. Cryptology ePrint Achive Report 2010/512, 2010. http://eprint.iacr.org/2010/512.

[14] 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.

[15] 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.

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

[17] 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.

[18] D. Dachman-Soled, T. Malkin, M. Raykova, and M. Yung. Secure efficient multiparty computing of

multivariate polynomials and applications. In *Applied Cryptography and Network Security (ACNS)*, pages 130–146, 2011.

[19] 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.

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

[21] 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.

[22] 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.

[23] 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.

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

[25] 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.

[26] 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.

[27] 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.

[28] 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.

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

[30] 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.

[31] 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.

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

[33] 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.

[34] 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.

[35] 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.

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

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

[38] 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.

[39] Y. Huang, D. Evans, and J. Katz. Private set intersection: Are garbled circuits better than custom protocols? In *Network & Distributed System Security Symposium (NDSS)*, 2012.

[40] 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.

[41] 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.

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

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

[44] H. T. Kung and P. Lehman. Systolic (VLSI) arrays for relational database operations. In *ACM SIGMOD International Conference on Management of Data*, pages 105–116, 1980.

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

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

[47] 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.

[48] 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.

[49] 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.

[50] 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.

[51] 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.

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

[53] A. K. Sood and M. Abdelguerfi. Parallel and pipelined processing of some relational algebra operations. *International Journal of Electronics Theoretical and Experimental*, 59(4):477–482, 1985.

[54] T. Toft. Sub-linear, secure comparison with two non-colluding parties. In *Public Key Cryptography (PKC)*, pages 174–191, 2011.

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

[56] G. Wang, T. Luo, M. Goodrich, W. Du, and Z. Zhu. Bureaucratic protocols for secure two-party sorting, selection, and permuting. In *ACM Symposium on Information, Computer and Communications Security (ASIACCS)*, pages 226–237, 2010.

[57] B. Zhang. Generic constant-round oblivious sorting algorithm for MPC. In *International Conference on Provable Security (ProvSec)*, pages 240–256, 2011.