Fast Private Set Operations with SEPIA

Dilip Many
dmany@tik.ee.ethz.ch

Martin Burkhart
burkhart@tik.ee.ethz.ch

Xenofontas Dimitropoulos
fontas@tik.ee.ethz.ch

TIK-Report No. 345
Communication Systems Group
ETH Zurich, Switzerland
1 March 2012

ABSTRACT

Private set operations allow correlation of sensitive data from multiple data owners. Although intensely researched, current solutions still exhibit limited scalability in terms of the supported maximum set size and number of sets. To address these issues, we propose a new approach to private set operations based on a combination of efficient secure multiparty computation and bloom filters, a space-efficient probabilistic data structure for set representation. We design, implement and evaluate protocols for counting and non-counting set intersection, set union, threshold set union, weighted set intersection, and set cardinality estimation. Evaluation in realistic settings shows that our protocols are between twenty times and several orders of magnitudes faster than the state-of-the-art.

Keywords

secure multiparty computation, set operations, bloom filters, privacy, collaboration

1. INTRODUCTION

Private set operations address the difficult problem of correlating sensitive information from different data owners. Possible application scenarios are very diverse, including, for instance, airlines checking passengers against a private no-fly list or credit card companies interested in finding a common list of customers likely to default without revealing their entire customer base. Recently, privacy-preserving data correlation has also been applied successfully to network security [6, 30, 21, 34]. In fact, set operations support many use cases in network security. For instance, distributed blacklists can be built from set union operations; set intersection allows the identification of common attackers and alerts; and sudden changes in the cardinality of active port or IP address sets are indicative for network anomalies and ongoing attacks [24]. While the underlying network data are often sensitive and subject to data protection legislation, they are also voluminous and correlation results must be available promptly in order to be useful. Internet-scale aggregation, in particular, poses new challenges to private set operations in terms of scalability. Potentially, millions of elements must be correlated across hundreds of sites contributing data within few minutes.

Up to today, the problem of building private set operations has attracted a lot of attention in the literature (e.g., [13, 22, 11, 1, 14, 31, 9]). However, existing solutions have a number of shortcomings. Many solutions are designed for two parties only, e.g., a client matching its set against a server's set [11, 33, 17, 18]. Moreover, most approaches [13, 14, 10, 25, 19, 9, 27] focus on a few operations and do not provide a complete suite of composable set operations. Most importantly, available solutions do not scale to the set sizes required in many practical scenarios. From the few protocols that have been actually implemented, the fastest set union takes 25 hours to aggregate 10 sets with 400 elements [28]. For set intersection, aggregation of only two sets with 65,000 elements still requires 20 minutes [20].

In this paper we design and implement a suite of private set operations that have short running times even with large sets and many parties. Our approach combines generic secure multiparty computation (MPC) with bloom filters [3, 4]. A bloom filter is a space-efficient data structure for representing sets that allows to perform set membership checks and to estimate the size of a represented set [26]. The MPC aggregation of bloom filters, either for set union or intersection, is computationally very efficient. Although the use of bloom filters introduces false positives (but no false negatives), the false-positive rate (FPR) decreases exponentially with the bloom filter size. Given the substantial performance improvements, we consider a small FPR acceptable for many applications. Network monitoring, in particular, naturally requires sampling or probabilistic methods to keep up with large traffic volumes (e.g., [12]).

We implement our set operations using the SEPIA library [7]. We chose SEPIA because it is optimized for large numbers of parallel operations and it has been recently shown to outperform similar frameworks. SEPIA is based on Shamir's secret sharing scheme [32] and supports private addition, multiplication, and comparison operations on shared secrets. Furthermore, it supports two roles: *Input peers* only provide input data to the computation process while *privacy peers* perform the computation. This allows supporting a large number of data providers without enlarging the set of computation nodes. SEPIA is secure in the honest-but-curious adversary model, i.e., computation is safe as long as the majority of privacy peers is honest.

We implement and evaluate protocols for counting and non-

counting set intersection, set union, threshold set union, weighted set intersection, and set cardinality estimation. As an example, union and intersection of 25 sets with 100,000 elements each, 9 computation nodes, and an FPR of 0.65%, requires only 75 seconds with our protocols. Remarkably, for union and intersection of many sets, the communication complexity is logarithmic in the number of input sets, while in other works it is at least linear (e.g. [22, 14, 25, 19, 8, 27]). Hence, large numbers of sets are easily supported. This greatly improves over current work and enables practical applications of private set operations in many domains.

The remainder of this paper is organized as follows. The following section provides background on MPC and bloom filters. Sections 3 and 4 introduce the set intersection and union operations. These basic operations serve as building blocks for the set cardinality, threshold set union and weighted set intersection operations presented in Sections 5 through 7. Section 8 improves the scalability of our protocols with respect to the number of participants, before we evaluate them in detail in Section 9. The paper discusses related work in Section 10 and is concluded in Section 11.

2. PRELIMINARIES

In this section we first introduce the SEPIA library and then provide a short overview of bloom filters. A summary of the notation, terminology and variables used is given in Tables 1 and 2.

Secure Multiparty Computation: The SEPIA library supports two basic roles: *input peers* (IN) and *privacy peers* (PP). The IPs are parties that want to correlate their private data. The PPs help the IPs by performing private computations on shared secrets. The level of privacy and the running time of basic MPC operations scale with the number of PPs. More PPs increase security but reduce performance and vice versa. Hence, the separation into IPs and PPs allows to tune the privacy-performance tradeoff independently of the number of participants with input data. In practice, a PP can be hosted together with an IP in the same network.

SEPIA uses Shamir's secret sharing scheme [32]. In order to share a secret in Shamir's scheme, a random polynomial over a prime field is generated and the secret *shares* correspond to different points on the polynomial. A secret is reconstructed from shares by interpolating the polynomial, e.g., using Lagrange interpolation. Private addition of secrets is done by simply adding the corresponding shares of a privacy peer. Private multiplication requires an intermediate step of synchronization and information exchange between the privacy peers [2, 16]. This also involves local computation that is more complex than for addition. Therefore to design efficient protocols it is desirable to reduce the number of multiplications. In Table 1 we introduce and describe the notation and terminology of the basic MPC operations we use. The *equal* and *lessThan* operations require much more resources in terms of running time and network bandwidth than the basic addition and multiplication operations. The required number of multiplications is logarithmic in the field size. To give an example, in a LAN setting with 5 PPs it is possible to perform 80,000 multiplications, 2,000 equality comparisons or 90 less-than comparisons per second. However, when the comparison operations are used wisely, they can help reduce the running time of protocols as we show in Section 8.

The protocols we introduce in this paper are compositions of additions and multiplications (comparisons are also compositions of additions and multiplications). Therefore they have the same security properties with the basic operations, which have been shown to be correct and information-theoretically secure [2].

Bloom Filters: We use the bloom filter data structure [3] to represent sets. A bloom filter for representing a set $S = \{x_1, x_2, \ldots, x_r\}$ of r elements is an array BF of s bits initially set to 0. We use the notation $BF(u)$ to refer to the position u in the bloom filter BF. The bloom filter uses k independent hash functions $h_1, \ldots, h_k$ with range $1, \ldots, s$. For each element $x \in S$, the bits at positions $h_i(x)$ are set to 1 for $1 \leq i \leq k$. To check whether an element y is a member of S, we simply test if $BF(h_i(y))$ equals 1 for all $1 \leq i \leq k$. As long as the bloom filter is not saturated, i.e., s is chosen sufficiently large to represent all elements, the total number of non-zero positions allows to accurately estimate $|S|$ [26]. Counting Bloom Filters (CBF) are a generalization of bloom filters, which use integer arrays instead of bit arrays. Thus, CBFs allow to represent *multisets*, in which each element can be represented more than once. When inserting elements, if a position $h_i(x)$ already has a count that is smaller than the one of x it is replaced by x's count. The count of an element in a CBF is retrieved by taking the minimum count of all positions an element hashes to.

Bloom filters introduce *false positives*, i.e., an element might seem present although it was never inserted. By increasing the size of the array this probability can be decreased for the price of increased space (and communication as we shall see later on) requirements. The FPR of a bloom filter is [4]:

$$FPR = \left(1 - \left(1 - \frac{1}{s}\right)^{kr}\right)^k \quad (1)$$

where s is the size of the bloom filter, k the number of hash functions and r the number of inserted elements. For CBFs, the probability that an element appears more often than it was inserted is also given by Equation 1.

The number of hash functions should be set according to the formula $k = ln(2) \cdot r/s$ derived in [4] to minimize the FPR for given s and r. A general rule of thumb is to set the size to $s = 5r \cdot (-\log_{10}(FPR))$. As an example, if we want to store 10^5 elements with an FPR of 10^{-3}, the size s should be $1.5 \cdot 10^6$. For an FPR of 10^{-4}, the number of required positions is only increased by 33% to $s = 2 \cdot 10^6$.

Note that while bloom filters allow to efficiently evaluating membership queries, they cannot be used to enumerate the represented set elements. This is not a concern when a bloom filter is used to check online, for example in a network intrusion detection system, if the elements of a stream belong to a set.

3. SET INTERSECTION

In order to intersect their sets, each input peer i first locally computes a binary BF_i or a counting CF_i bloom filter, repre-

Syntax	Output	Description
$share(s)$	$[s]$	A secret value s held by an IP is split up in m shares s_i. Each s_i is then distributed to PP i. The ensemble of all distributed shares $\{s_1, s_2, \cdots, s_m\}$ is called a *sharing* of s and denoted by $[s]$.
$reconstruct([s])$	s	The individual shares of a sharing $[s]$ are combined to reconstruct the secret value s.
$[a] + [b]$, $[a] + b$	$[a + b]$	Adds two sharings (or a sharing and a public value) to get a sharing of the sum. This also includes subtraction ('-').
$[a] * [b]$, $[a] * b$	$[a * b]$	Multiplies two sharings (or a sharing and a public value) to get a sharing of the product.
$equal([a], [b])$, $equal([a], b)$	if $(a == b)$ then [1] else [0]	Two sharings (or a sharing and a public value) are compared for equality. The output remains secret.
$lessThan([a], [b])$, $lessThan([a], b)$	if $(a < b)$ then [1] else [0]	Two sharings (or a sharing and a public value) are compared for size. The output remains secret.

Table 1: Notation and terminology for basic MPC operations.

n	the number of input peers with a set each
m	the number of privacy peers
s	size of the bloom filter (i.e., number of positions)
k	number of hash functions per element
r	number of inserted distinct elements
p	the field size used in secret sharing
l	the required number $\lceil \log_2(p) \rceil$ of bits to represent p
BF	a binary bloom filter
CF	a counting bloom filter

Table 2: Summary of main variables.

senting its input set. Each array position of the bloom filter is then shared among the privacy peers. The privacy peers then compute the bloom filter representing the intersection set position-wise. For binary bloom filters, they perform the logical AND for all positions u with $1 \leq u \leq s$:

$$[BF_\wedge(u)] := [BF_1(u)] \wedge [BF_2(u)] \wedge \ldots \wedge [BF_n(u)]. \quad (2)$$

We can implement a private logical AND simply with a private multiplication: $[Bit1 \wedge Bit2] = [Bit1] * [Bit2]$. All positions of $[BF_\wedge]$ are then reconstructed and are sent back to the input peers, unless they are used for further private computations as part of the protocols in the following sections, in which case they remain secret.

For the multiset case the privacy peers perform the following computation:

$$[CF_\wedge(u)] := min(min(min([CF_1(u)], [CF_2(u)]), [CF_3(u)]), \ldots, [CF_n(u)]) \quad (3)$$

Again the minimum is computed position-wise over all bloom filter positions. We compute the minimum of two secret sharings $[a]$ and $[b]$ using SEPIA's *lessThan* operation as follows:

$$min([a], [b]) := [a] * lessThan([a], [b]) + [b] * (1 - lessThan([a], [b]))$$

Rather than computing the minimum operations sequentially as shown in Equation 3, we compare the n input numbers in a tree-like fashion, which enables to parallelize minimum operations.

The aggregate binary bloom filter has a slightly higher FPR than a bloom filter that would be constructed directly from the final intersection set. Intuitively, bits in the input bloom filters BF_i set to 1 by elements that are not part of the final intersection can survive in the aggregate bloom filter $BF_\wedge$ due to collisions with bits from other elements.

In the case of counting bloom filters the count retrieved for an element x is overestimated if all its positions $h_i(x)$ collide with elements having a higher count. We provide a detailed analysis of the FPR in bloom filters representing intersected sets in Section 5.2. For both binary and counting bloom filters, the FPR of $BF_\wedge$ or $CF_\wedge$ is lower than the FPR in any input bloom filter (BF_i and CF_i). Therefore, if the bloom filters are dimensioned so that the FPR is acceptable for representing the input sets, then FPR is also acceptable for the intersection.

4. SET UNION

The computation of set union is similar to the computation of set intersection. First, the input peers compute the binary bloom filters BF_i representing their sets and share them among the privacy peers. The privacy peers then compute the bloom filter representing the unified set for all u with $1 \leq u \leq s$ as follows:

$$[BF_\vee(u)] := [BF_1(u)] \vee [BF_2(u)] \vee \ldots \vee [BF_n(u)] \quad (4)$$

We compute the logical OR ($\vee$) using a private addition and a private multiplication operation:

$$[Bit1 \vee Bit2] := [Bit1] + [Bit2] - [Bit1] * [Bit2].$$

In the end, the items of $[BF_\vee]$ are reconstructed and sent to the input peers, unless they are used for further computations, in which case they remain secret.

For the multiset case, we use simply an addition instead of the OR operation:

$$[CF_\vee(u)] := [CF_1(u)] + [CF_2(u)] + \ldots + [CF_n(u)] \quad (5)$$

Unlike the set intersection operation, the set union does not affect the FPR. That is, the FPR of $BF_\vee$ or of $CF_\vee$ is equal to the FPR of a bloom filter that is constructed from the final union directly:

$$FPR_\vee = \left(1 - \left(1 - \frac{1}{s}\right)^{krn}\right)^k \quad (6)$$

5. SET CARDINALITY

The number of true bits in a binary bloom filter is [26]:

$$\sum_{u=1}^{s} BF(u) = \tau \approx \hat{\tau}(r) = s\left(1 - \left(1 - \frac{1}{s}\right)^{kr}\right) \quad (7)$$

By denoting the estimate of the number of inserted elements r as $\hat{r}$ and solving the equation for it we get:

$$r \approx \hat{r}(\tau) = \frac{\ln(1 - \frac{\tau}{s})}{k \ln(1 - \frac{1}{s})} \quad (8)$$

For counting bloom filters we can compute the number of elements inserted exactly, assuming that there was no overflow during the insertion phase, with the equation:

$$r = \frac{\tau}{k} \quad (9)$$

Observe that to estimate the set cardinality of a bloom filter or of a counting bloom filter using the last two equations, it is necessary to first sum the values (bits or integers) in the array of a filter.

5.1 Cardinality of Union

Private set cardinality for union operations is straightforward. We first compute the union of input bloom filters as discussed in Section 4, but instead of reconstructing the aggregate bloom filter, we compute the sum of the values in a filter array using SEPIA's private addition protocol. We then reconstruct the sum from which input peers compute the cardinality of the union of their private input sets using Equations 8 and 9 in the case of binary and counting bloom filters, respectively. One could also compute Equations 8 and 9 in MPC and reconstruct the cardinality instead of the sum. However, computing logarithms in MPC or dividing in MPC using algebraic circuits is not trivial. In addition, since the sum can be directly computed from the set cardinality, disclosing the former does not reveal any additional information.

5.2 Cardinality of Intersection

Private set cardinality for intersection operations is more complex. For bloom filters resulting from intersection operations we cannot directly use the equations above to estimate the cardinality of the intersection, because an aggregate bloom filter after a position-wise AND may contain additional falsely set bits that do not correspond to elements of the intersection. We call these *false bits* due to aggregation. In the following we describe for the case of binary bloom filters how to estimate the number of false bits and the cardinality of the intersection.

The work by Papapetrou *et al.* [26] provides equations to estimate the set cardinality of the intersection of *two* binary bloom filters. We extend the calculation to an arbitrary number n of intersected sets. Let $BF_\wedge$ be the intersection computed with the sets represented by bloom filters while $BF_\cap$ denotes the intersection computed from the exact sets. Furthermore, we define SET_{BF} to be the set of array indices of true bits in BF. The set $SET_{BF_\wedge}$ is a superset of the set $SET_{BF_\cap}$. Then, $e = |SET_{BF_\wedge}| - |SET_{BF_\cap}|$ is the number of false bits. The probability that a bit is set in all sets $SET_{BF_i} \backslash SET_{BF_\cap}$ for $1 \leq i \leq n$ and therefore is a false bit in $SET_{BF_\wedge}$ is:

$$P(BF_\wedge(u) = \neg BF_\cap(u)) = \prod_{i=1}^{n} \frac{|SET_{BF_i}| - |SET_{BF_\cap}|}{s - |SET_{BF_\cap}|} = \prod_{i=1}^{n} \frac{\tau_i - \tau_\cap}{s - \tau_\cap} \quad (10)$$

Let $\tau_\cap = |SET_{BF_\cap}|$, $\tau_\wedge = |SET_{BF_\wedge}|$, and $\hat{\tau_\wedge} = \tau_\cap + \hat{e}$. Then, an estimate of e is $\hat{e} = (s - \tau_\cap) * P(BF_\wedge(u) = \neg BF_\cap(u))$. We replace all τ_i with a fixed τ_q (to be determined later on), which leads to the equation:

The problem with this estimate is that each input peer only knows the cardinality of his input set. Therefore

$$\hat{\tau_\wedge} = \tau_\cap + \frac{(\tau_q - \tau_\cap)^n}{(s - \tau_\cap)^{n-1}} \quad (11)$$

Inserting 7 we get:

$$\hat{\tau_\wedge}(r_\cap, \tau_q) = s - s\left(1 - \frac{1}{s}\right)^{kr_\cap} + \left(s\left(1 - \frac{1}{s}\right)^{kr_\cap}\right)^{1-n} \left(\tau_q - s\left(1 - \left(1 - \frac{1}{s}\right)^{kr_\cap}\right)\right)^n \quad (12)$$

From the last equation we cannot derive a closed form for $\hat{r_\cap}(\tau_\wedge, \tau_q)$. However, we can use numerical methods to find a $\hat{r_\cap}$ s.t. $f(r_\cap; \tau_\wedge, \tau_q) = \hat{\tau_\wedge}(r_\cap, \tau_q) - \tau_\wedge = 0$. As the above function is monotone we use Newton's Method to find $\hat{r_\cap}$. In our experiments, Newton's Method had negligible overhead compared to MPC.

We tested the above formula with different numbers of input sets, distributions of elements and choices of τ_q. We evaluated selecting τ_q as the lowest number of true bits in any input set, as the average number of true bits, and as

the number of true bits in the intersection filter. In our experiments we observed that the error of the estimate decreases quickly as the number of input sets increases, independently of the distribution of elements and of the choice of τ_q. In most cases, the average number of true bits of all input sets performed well. Computing this value in MPC is fast and does not disclose any additional information. The interpolation can then be done by the input peers locally. Alternatively the input peers could also use the number of true bits of their own input sets without disclosing any additional information. We leave for future work the derivation of a formal bound on the precision of τ_q.

6. THRESHOLD SET UNION

Let $CF_\vee$ be the union of the counting bloom filters $CF_1, ..., CF_n$ as described in Section 4. Given a threshold d we want to compute the elements that occur at least d times in the union. This *threshold set union* can be computed as:

$$\begin{aligned}[CF_{TU}(u)] &= ThresholdUnion([CF_1(u)], \ldots, [CF_n(u)], d) \\ &= 1 - lessThan([CF_\vee(u)], d) \end{aligned} \quad (13)$$

where the private *lessThan* operation is computed positionwise over $1 \leq u \leq s$. Based on this, a threshold set union with counts reconstructed can be computed using a private multiplication operation as:

$$[CF_{TUC}(u)] = [CF_{TU}(u)] * [CF_\vee(u)] \quad (14)$$

This second threshold set union operation maintains the counts of the elements that are in the threshold union. It computes what was defined as Private Over-Threshold Set-Union in [22]. Unless $[CF_{TU}]$ or $[CF_{TUC}]$ are used for further computations and shall remain secret, the positions are reconstructed and sent to the input peers.

Threshold set union operations may introduce falsely set bits (in CF_{TU}) or counters (in CF_{TUC}) similarly to the set union operation discussed in Section 4. But unlike in the multiset union the bits or counts are only falsely set if the count is above the threshold. Therefore the probability is lower than the FPR of multiset union.

7. WEIGHTED SET INTERSECTION

The threshold set union allows filtering elements by a threshold on their count. In certain scenarios, however, it is useful to also have a threshold on the number of sets that contain a certain element. For instance, when correlating security alerts across different sites, the participants might not be interested in events that are restricted to a single site. They might agree on a disclosure policy that only reconstructs elements, which have an aggregate count above a certain threshold *and* are visible on a minimum number of sites. That is, we are looking for a protocol that supports two thresholds. The *weighted set intersection* protocol finds the elements that are contained in at least t_c sets and that have an aggregate count of at least t_w.

A deterministic weighted set intersection protocol was introduced in [7], called *event correlation protocol*. However, the event correlation protocol has limited scalability regarding the number of elements (or equivalently events) per input peer. In particular, correlating 25 sets with only 30 elements per set and 9 PPs takes already more than 200 seconds. Moreover, the computation time of the event correlation protocol scales quadratically with the number of elements.

Using the protocols we introduced in the previous sections, we build a much more efficient protocol for weighted set intersection. Each input peer has a set of tuples, where each tuple has a key k_i and an integer weight w_i. The weighted set intersection contains the keys that appear in at least t_c different sets and that have an aggregate weight of at least t_w:

$$WSI = \left\{ k_i \left| \sum_{\substack{j \\ k_j = k_i}} 1 \geq t_c \bigwedge \sum_{\substack{j \\ k_j = k_i}} w_j \geq t_w \right. \right\} \quad (15)$$

The input sets can be represented by two bloom filters. The key filter is a counting bloom filter created by inserting each key with count 1. The weight filter is a counting bloom filter into which each key is inserted with its weight as count. The weighted set intersection protocol proceeds with the following steps:

1. Each IP i creates a bloom filter for keys CFK_i and a counting bloom filter for weights CFW_i.
2. Each IP shares $CFK_i(u)$ and $CFW_i(u)$, where $1 \leq u \leq s$, among the privacy peers.
3. The privacy peers compute the threshold set union of the key filters:

   $$CFK_{TU}(u) = ThresholdUnion(CFK_1(u), \ldots, CFK_n(u), t_c)$$

   and of the weight filters:

   $$CFW_{TU}(u) = ThresholdUnion(CFW_1(u), \ldots, CFW_n(u), t_w)$$

   For the weight filters, instead of the standard *ThresholdUnion* (Equation 13) we can also use Equation 14 to reconstruct the aggregate weights of the elements in the weighted set intersection.
4. The privacy peers compute $CBF_{WSI}(u) = CFK_{TU}(u) * CFW_{TU}(u)$ and reconstruct the result.

This protocol nicely demonstrates a key benefit of our approach. Since all set operations are built with standard MPC operations, it is possible to combine set operations with generic MPC operations and build powerful composite protocols.

8. SCALING TO LARGE NUMBERS OF SETS

For the operations on binary bloom filters introduced in the previous sections it is often necessary to compute the AND ($\wedge$) and OR ($\vee$) of many values. With n input sets, these operations require $n-1$ MPC multiplications per bloom filter

position. MPC multiplications require the PPs to synchronize intermediate values and hence are much more costly than MPC additions, which only require local computation by the PPs.

Fortunately, the number of expensive multiplications can be reduced for large n. The optimized operations use a combination of additions and an equality operation. Let $l = \lceil \log_2(p) \rceil$ be the number of bits required to represent p and $q \leq l$ be the number of bits set to 1 in the binary representation of the group order p. The equality operation requires $l + q - 2$ multiplications [7]. Therefore if $l + q - 2 < n - 1$ we use the more efficient operations described by Equations 16 and 17.

$$fastAND(x_1, x_2, \ldots, x_n) := Equal\left(\sum_{i=1}^{n} 1 - x_i, 0\right) \quad (16)$$

$$fastOR(x_1, x_2, \ldots, x_n) := 1 - Equal\left(\sum_{i=1}^{n} x_i, 0\right) \quad (17)$$

These operations have the big advantage that their running time mostly depends on the speed of the equal operation which depends only on the choice of the group order p. The group order p has to be chosen s.t. $p > max(m + 1, n - 1)$. When $l + q - 2 < n - 1$ the running time of the operations only depends logarithmically on the number n of sets. The additions and subtractions can be done locally and the time required for that is negligible. In addition, given a fixed group order p, which is the case in practice, the running time of the operations is *constant* in the number of sets. Finally, note that since l, q, and n are known in the beginning of a computation, we can automatically select the faster protocol for AND and OR operations.

With these optimizations the protocols' multiplication requirements are listed in Table 3.

9. PERFORMANCE EVALUATION

In this section we evaluate the performance of our set operations. We measure the average total protocol running time, the volume of transmitted data, and the memory usage. The CPU time is the total time it takes a privacy peer to process the received data and prepare the data to be sent. Communication time is the total time it takes a privacy peer to send and receive data. We do not evaluate the threshold set union protocol separately as the weighted set intersection protocol runs the former twice and therefore it reflects its performance accurately. The set cardinality protocols were not evaluated as the computation time is practically negligible. Compared to set intersection and union, our set cardinality protocols require only an addition and local computation, which is practically negligible.

Parameters: We vary the number of input peers between 5 and 25 and the number of privacy peers between 3 and 9. Recall that the number of input peers n is equal to the number of input sets. The computation complexity of our protocols is linear with the bloom filter size. Assuming that in many practical scenarios the running time is upper-bounded by a few minutes, we select $s = 2^{20} = 1,048,576$ for the binary bloom filter of union and intersection operations. The protocols for weighted set and multiset intersection require $lessThan$ comparisons, which are much more costly than basic addition and multiplication operations [7]. We select a size of $s = 2^{14} = 16,384$ for the counting bloom filters of these protocols. We set the number of hash functions to $k = ln(2) \cdot r/s$ as suggested in [4]. The bloom filter size s determines the number of supported elements and the corresponding FPR (Equation (1)). As an example, we provide the FPR with 100,000 and 1,500 inserted elements in Table 4. For each configuration, the protocols were run 10 times. The error bars in the plots show the standard deviation of the running time.

Setup: Our evaluation experiments were run on systems with two AMD Opteron 2218 Dualcore 2.6GHz CPUs and 8 GB RAM that were running Debian and were interconnected by a 1 Gb/s LAN. The Java version used was OpenJDK Runtime Environment (IcedTea6 1.8.7) (6b18-1.8.7-2 squeeze1) and OpenJDK 64-Bit Server VM (build 14.0-b16, mixed mode). Each input and privacy peer was run on a separate system. In wide-area network (WAN) settings with lower network bandwidth the protocols will, of course, run slower. The influence of lower network speeds on the performance of the basic operations of SEPIA was measured in [5]. For the bottleneck multiplication operation, the share of total running time used for communication is 21% with 1Gb/s connections. With 100Mb/s connections, the share of communication time goes up to 32%. Thus, the major part of running time is due to local computation and does not significantly depend on link speeds.

9.1 Set Intersection

As shown in Fig. 1, the average running time of the binary set intersection with 10 input peers is larger than with 5 input peers, while for $n > 10$ the running time is almost constant. The reason for this change is that for $n > 10$, the fastAND is automatically used. We do not observe a logarithmic scaling behavior in the number of sets as we use a fixed field size p for all configurations. Therefore as discussed in Section 8, the cost for intersecting additional sets with the fastAND is negligible. For the multiset intersection, the average total running time and bandwidth usage always depends linearly on the number of input peers. As shown in Fig. 1 and Table 4 the binary set version is much faster than the multiset version.

Comparing the performance of our protocols to related work requires scaling down our parameters. For instance, if we take our performance results for 5 sets with 100,000 elements and 3 privacy peers and scale it down to two sets of 5000 elements, which corresponds to the currently fastest implementation [11], our implementation is about twenty times faster.

9.2 Set Union

Fig. 2 shows how the CPU running time scales with n and m for the binary (left) and the multiset (right) union. The scaling behavior of the running time of the set union protocol is similar to the set intersection protocol. For the multiset union, there is no distinctive dependency on either the num-

Operation	Sets	Multisets
Intersection	$(n-1)*s$ if $l+q-2 \geq n-1$ $O(log_2(n)*s)$ if $l+q-2 < n-1$	$O((n-1)*l*s)$
Union	$(n-1)*s$ if $l+q-2 \geq n-1$ $O(log_2(n)*s)$ if $l+q-2 < n-1$	0
Cardinality	1	1
Threshold Set Union	-	$s*l$ without counts $s*(l+1)$ reconstructing counts
Weighted Set Intersection	-	$2*s*l+s$ without counts $2*(s*l+s)$ reconstructing counts

Table 3: Required multiplications for our set operations.

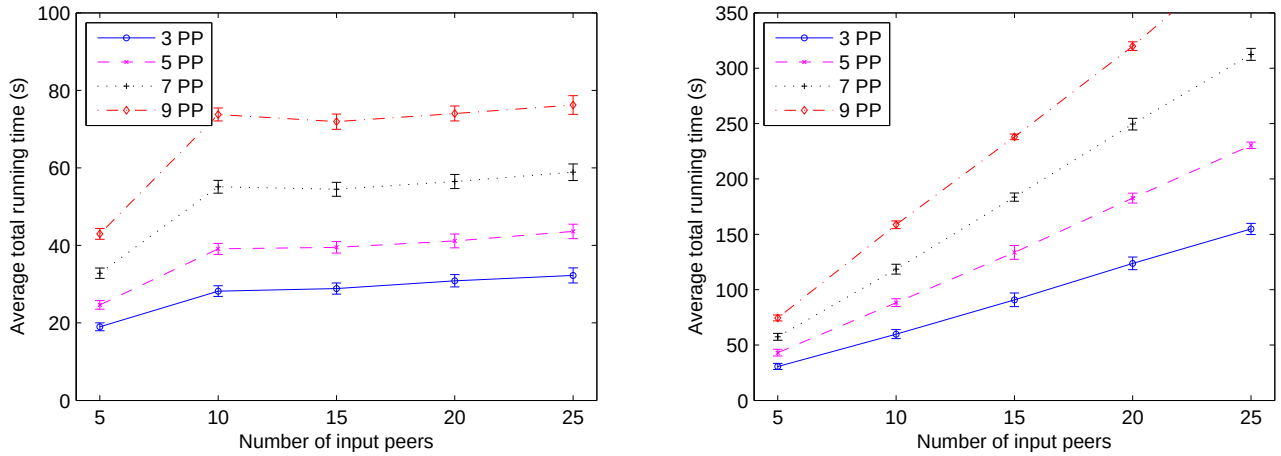

Figure 1: Average total set intersection protocol running time for binary bloom filter with size of 1,048,576 (left) and for multisets (right) with a filter size of 16,384. (PP = Privacy Peer).

ber of input or privacy peers. This comes from the fact that the multiset union only applies MPC addition, which is a very cheap local operation and does not require communication (besides the initial sharing and the final reconstruction). The curve for the communication time (not shown) for the binary union has the same shape with the curve for the CPU time. The communication time for mutliset union was always almost constant in our experiments close to two seconds.

Table 5 compares performance parameters of the set union protocol in the binary and in the multiset case. Compared to the corresponding statistics for the set intersection protocol shown in Table 4, in the union protocol the multiset version is faster than the binary version. Besides, the average total running time, the CPU time, the communication time and the traffic volume for binary bloom filters are very similar to the corresponding numbers for the intersection protocols.

The fastest set union implementation we are aware of takes 25 hours to compute the union of 10 sets of 400 elements [28]. However, this implementation cannot compute the union of sets as large as ours as the running time of their protocols is quadratic in the number of sets and the set size. As our protocol can process far more data in a much shorter time, our set union is several orders of magnitude faster.

9.3 Weighted Set Intersection

Fig. 3 shows the average total running time of the weighted set intersection protocol (left) and the event correlation protocol (right) presented in [7]. The event correlation protocol uses sets of size 30, while the weighted set intersection protocol uses bloom filters of size 16,384, which can represent much larger sets, for example, of 1,500 elements with a 0.5% FPR.

The total as well as the average CPU and communication time of our protocol are all approximately constant for varying numbers of input peers because the operation is essentially independent of the number of input peers. The number of input peers only influences the traffic volume of input data that has to be submitted to the privacy peers and the traffic volume of results sent back. The event correlation protocol in [7], which served as an inspiration for our weighted set intersection protocol, scales quadratically both in the number of input peers and the set size. The running time of our protocol is independent of the number of input peers n. Furthermore, it only scales linearly with the set size.

Table 6 compares the performance of the weighted set intersection protocol with and without reconstructing the weights. As it can be seen from the numbers the effect of the reconstruction is almost negligible.

Parameter	Set intersection	Multiset intersection
Bloom filter size	1,048,576	16,384
Field size	101	101
# Elements	100,000	1500
False positive rate	0.65%	0.5%
Avg. runtime, 3 PP, 5 IP (CPU/Com/Total)	(4.4/4.5/19.0)s	(11.2/9.1/30.7)s
Avg. runtime, 9 PP, 25 IP (CPU/Com/Total)	(44.2/11.0/76.2)s	(184.4/165.8/411.6)s
Per PP traffic at 9 PP, 25 IP	352.3 MB	approx. 1 GB
Per IP traffic at 9 PP, 25 IP	26.9 MB	approx. 1 MB

Table 4: Summary of main configuration and performance parameters for set and multiset intersection.

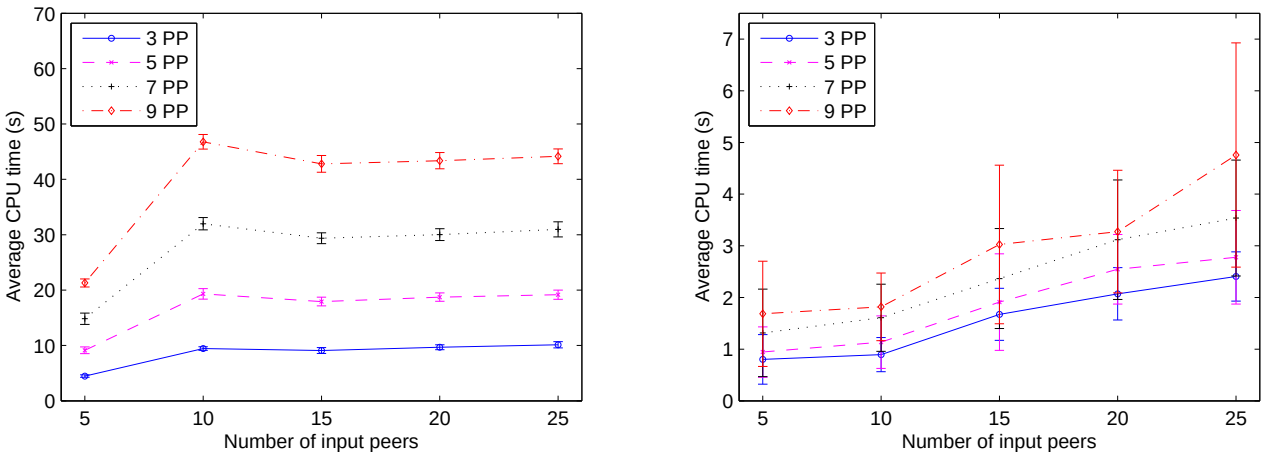

Figure 2: Average CPU time for the execution of the set union protocol with binary (left) and counting (right) bloom filters of size 1,048,576.

10. RELATED WORK

Several works [22, 15, 19, 14, 18] describe set operations based on encrypted polynomials. In this approach, set elements are represented by the roots of a polynomial. The coefficients of all the polynomials are encrypted using a homomorphic encryption scheme. Due to the encryption these operations guarantee only computational security. Certain schemes allow for malicious players by incorporating zero-knowledge proofs [22, 14, 19, 18]. The performance of [15] is shown in Table 7, where we illustrate the performance of related implemented set operations as evaluated by their authors. Cheon *et al.* [8] also use encryption and polynomials for set representation but with a twist. The authors evaluate the polynomial at fixed points given by an index set achieving a protocol that scales quasi-linearly in the set size.

Furthermore, certain works [11, 9, 17] have developed interesting custom cryptographic protocols for secure set operations. De Cristofaro *et al.* [11, 9] describe protocols with linear complexity to compute the intersection and the cardinality of the intersection of two sets. The performance of these two protocols is shown in Table 7. Ye *et al.* [33] use polynomials for set representation and combine secret sharing with encryption to compute the union of two sets.

Another set of works [25, 31, 27] combine the polynomial representation of sets with Shamir's Secret Sharing scheme [32], which provides information-theoretic security, to compute set intersection operations. However, the scaling behavior of these schemes is moderate: The complexity is $O(n^3 * r^2)$ for [25] (with passive adversaries) and $O(n * r^2)$ for [31]. In contrast, the scaling behavior of our protocols is summarized in Table 3.

Besides, Lai *et al.* [23] propose to do set operations with sets represented by bloom filters. But the privacy of the sets in this work is only protected by the false positive rate of the filter and by splitting up the filter into segments revealing only one segment to each other player. This means that privacy has to be bought with a reduction of precision and t players will still learn t segments of each filter.

In [29] the authors describe GCR, an additive secret sharing scheme and mention the potential use of bloom filters for private set operations. However, the GCR scheme allows only for the single case of union on counting bloom filters.

In the Table 7 we summarize key configuration parameters and results as reported by their authors from the evaluation of related implemented set operations. We did not attempt to scale the configuration parameters to a consistent scenario as they use different techniques which scale differently. Our protocols are much faster in some cases by several orders of magnitude than previous set operations.

Parameter	Set Union	Multiset Union
Bloom filter size	1,048,576	1,048,576
Field size	101	1,107,296,257
# Elements	100,000	100,000
False positive rate	0.65%	0.65%
Avg. runtime at 3 PP, 5 IP (CPU/Com/Total)	(4.5/4.3/19.1)s	(0.8/1.4/12.9)s
Avg. runtime at 9 PP, 25 IP (CPU/Com/Total)	(44.2/10.4/75.9)s	(4.8/2.3/33.7)s
Per PP traffic at 9 PP, 25 IP	352.3 MB	239.2 MB
Per IP traffic at 9 PP, 25 IP	26.9 MB	56.2 MB

Table 5: Summary of main configuration and performance parameters for set and multiset union.

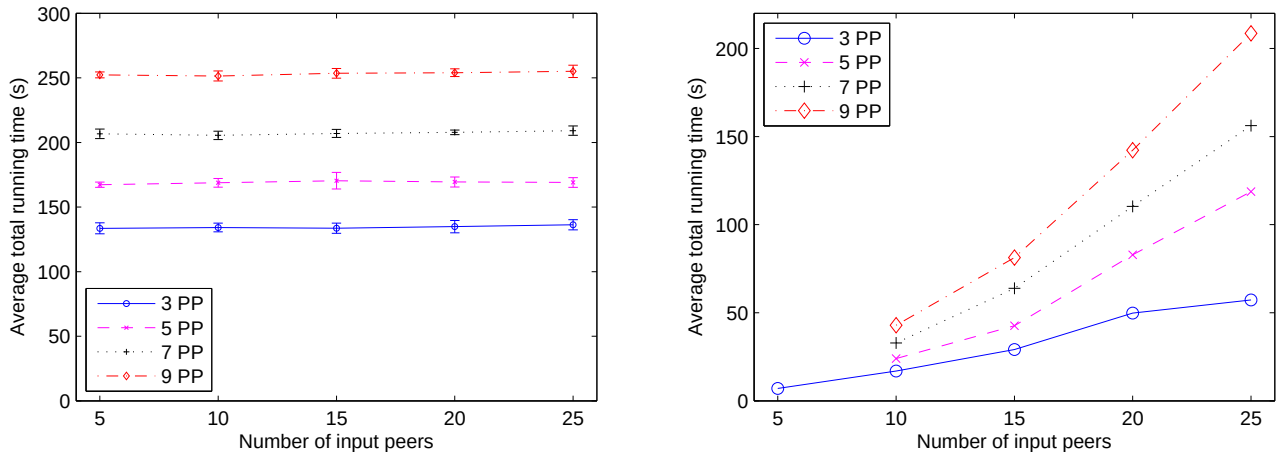

Figure 3: Total time of the weighted set intersection protocol with a bloom-filter size of 16,384 (left) and of the event correlation protocol of the USENIX 2010 paper [7] with sets of size 30 (right).

11. CONCLUSIONS

In this work we designed, implemented and evaluated a collection of fast and composable set operations providing information-theoretic security based on Shamir's Secret Sharing scheme. The operations were optimized to work efficiently for multiple parties and large sets. They can be combined with generic MPC operations to build complex composite protocols, as illustrated by our WSI protocol. Remarkably, for union and intersection of many sets, the communication complexity is logarithmic in the number of input sets. Compared to prior works our set intersection implementation is about twenty times faster than the current state-of-the-art and our set union implementation several orders of magnitude.

12. ACKNOWLEDGMENTS

We want to thank Manuel Widmer for his contributions to this work. This work was supported by DEMONS, a research project supported by the European Commission under its 7th Framework Program (contract no. 257315).

13. REFERENCES

[1] G. Ateniese, E. De Cristofaro, and G. Tsudik. (if) size matters: Size-hiding private set intersection. In D. Catalano, N. Fazio, R. Gennaro, and A. Nicolosi, editors, *Public Key Cryptography Ű PKC 2011*, volume 6571 of *Lecture Notes in Computer Science*, pages 156–173. Springer Berlin / Heidelberg, 2011.

[2] M. Ben-Or, S. Goldwasser, and A. Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation. In *ACM symposium on Theory of computing (STOC)*, 1988.

[3] B. H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Commun. ACM*, 13:422–426, July 1970.

[4] A. Broder and M. Mitzenmacher. Network applications of bloom filters: A survey. *Internet Mathematics*, 1(4):485–509, 2004.

[5] M. Burkhart. *Enabling Collaborative Network Security with Privacy-Preserving Data Aggregation*. PhD thesis, ETH Zurich, July 2011.

[6] M. Burkhart and X. Dimitropoulos. Privacy-Preserving Distributed Network Troubleshooting - Bridging the Gap between Theory and Practice. *ACM Transactions on Information and System Security (TISSEC)*, 14(4), Dec. 2011.

[7] M. Burkhart, M. Strasser, D. Many, and X. Dimitropoulos. SEPIA: Privacy-Preserving Aggregation of Multi-Domain Network Events and Statistics. In *19th USENIX Security Symposium*, Washington, DC, USA, Aug. 2010.

[8] J. H. Cheon, S. Jarecki, and J. H. Seo. Multi-party privacy-preserving set intersection with quasi-linear complexity. Cryptology ePrint Archive, Report 2010/512, 2010.

[9] E. D. Cristofaro, P. Gasti, and G. Tsudik. Fast and

Parameter	W/o weights	With weights
Bloom filter size	16,384	16,384
Field size	1,107,296,257	1,107,296,257
# Elements	1500	1500
False positive rate	0.5%	0.5%
Avg. runtime, 3 PP, 5 IP (CPU/Com/Total)	(91.4/30.4/133.7)s	(89.7/32.1/133.5)s
Avg. runtime, 9 PP, 25 IP (CPU/Com/Total)	(145.3/76.7/257.3)s	(141.2/78.3/255.0)s
Per PP traffic at 9 PP, 25 IP	837.6 MB	798.0 MB
Per IP traffic at 9 PP, 25 IP	1.6 MB	1.6 MB

Table 6: Summary of main configuration and performance parameters for weighted set intersection with and w/o weight reconstruction.

Reference	Year	Operation	Number of Sets	Set size	Running time
[15]	2006	set intersection	2	160	40s
[11]	2010	set intersection	2	5,000	1.4s (client) 4.7s (server)
[20]	2011	set intersection	2	65,536	20m
[9]	2011	set intersection cardinality	2	1,000	3s
[28]	2010	set union	10	400	25h

Table 7: Comparison of the running time reported in the literature for different implemented set operations.

private computation of set intersection cardinality. Cryptology ePrint Archive, Report 2011/141, 2011. `http://eprint.iacr.org/`.

[10] E. De Cristofaro, J. Kim, and G. Tsudik. Linear-complexity private set intersection protocols secure in malicious model. In M. Abe, editor, *Advances in Cryptology - ASIACRYPT 2010*, volume 6477 of *Lecture Notes in Computer Science*, pages 213–231. Springer Berlin / Heidelberg, 2010.

[11] E. De Cristofaro and G. Tsudik. Practical private set intersection protocols with linear complexity. In R. Sion, editor, *Financial Cryptography and Data Security*, volume 6052 of *Lecture Notes in Computer Science*, pages 143–159. Springer Berlin / Heidelberg, 2010.

[12] X. Dimitropoulos, P. Hurley, and A. Kind. Probabilistic lossy counting: an efficient algorithm for finding heavy hitters. *ACM SIGCOMM Computer Communication Review*, 38(1):5–5, 2008.

[13] M. Freedman, K. Nissim, and B. Pinkas. Efficient private matching and set intersection. In C. Cachin and J. Camenisch, editors, *Advances in Cryptology - EUROCRYPT 2004*, volume 3027 of *Lecture Notes in Computer Science*, pages 1–19. Springer Berlin / Heidelberg, 2004.

[14] K. Frikken. Privacy-preserving set union. In J. Katz and M. Yung, editors, *Applied Cryptography and Network Security*, volume 4521 of *Lecture Notes in Computer Science*, pages 237–252. Springer Berlin / Heidelberg, 2007.

[15] S. Garriss, M. Kaminsky, M. J. Freedman, B. Karp, D. Mazières, and H. Yu. Re: Reliable email. In *NSDI Š06: 3rd Symposium on Networked Systems Design & Implementation*, page 297Ű310, 2006.

[16] R. Gennaro, M. Rabin, and T. Rabin. Simplified VSS and fast-track multiparty computations with applications to threshold cryptography. In *7th annual ACM symposium on Principles of distributed computing (PODC)*, 1998.

[17] C. Hazay and Y. Lindell. Efficient protocols for set intersection and pattern matching with security against malicious and covert adversaries. In R. Canetti, editor, *Theory of Cryptography*, volume 4948 of *Lecture Notes in Computer Science*, pages 155–175. Springer Berlin / Heidelberg, 2008.

[18] C. Hazay and K. Nissim. Efficient set operations in the presence of malicious adversaries. In P. Nguyen and D. Pointcheval, editors, *Public Key Cryptography Ű PKC 2010*, volume 6056 of *Lecture Notes in Computer Science*, pages 312–331. Springer Berlin / Heidelberg, 2010.

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

[20] Y. Huang, D. Evans, J. Katz, and L. Malka. Faster secure two-party computation using garbled circuits. In *Proceedings of the 20th USENIX conference on Security*, SEC'11, pages 35–35, Berkeley, CA, USA, 2011. USENIX Association.

[21] S. Katti, B. Krishnamurthy, and D. Katabi. Collaborating against common enemies. In *Internet Measurement Conference (IMC)*, 2005.

[22] L. Kissner and D. Song. Privacy-preserving set operations. In V. Shoup, editor, *Advances in Cryptology Ű CRYPTO 2005*, volume 3621 of *Lecture Notes in Computer Science*, pages 241–257. Springer Berlin / Heidelberg, 2005.

[23] P. K. Y. Lai, S.-M. Yiu, K. P. Chow, C. F. Chong, and L. C. K. Hui. An efficient bloom filter based solution for multiparty private matching. In *Security and Management*, pages 286–292, 2006.

[24] A. Lakhina, M. Crovella, and C. Diot. Mining anomalies using traffic feature distributions. In *ACM SIGCOMM*, 2005.

[25] R. Li and C. Wu. An unconditionally secure protocol for multi-party set intersection. In J. Katz and M. Yung, editors, *Applied Cryptography and Network Security*, volume 4521 of *Lecture Notes in Computer Science*, pages 226–236. Springer Berlin / Heidelberg, 2007.

[26] O. Papapetrou, W. Siberski, and W. Nejdl. Cardinality estimation and dynamic length adaptation for bloom filters. *Distributed and Parallel Databases*, 28:119–156, 2010. 10.1007/s10619-010-7067-2.

[27] A. Patra, A. Choudhary, and C. Rangan. Information theoretically secure multi party set intersection re-visited. In M. Jacobson, V. Rijmen, and R. Safavi-Naini, editors, *Selected Areas in Cryptography*, volume 5867 of *Lecture Notes in Computer Science*, pages 71–91. Springer Berlin / Heidelberg, 2009.

[28] T. Raeder, M. Blanton, N. Chawla, and K. Frikken. Privacy-preserving network aggregation. In M. Zaki, J. Yu, B. Ravindran, and V. Pudi, editors, *Advances in Knowledge Discovery and Data Mining*, volume 6118 of *Lecture Notes in Computer Science*, pages 198–207. Springer Berlin / Heidelberg, 2010.

[29] F. Ricciato and M. Burkhart. Reduce to the max: A simple approach for massive-scale privacy-preserving collaborative network measurements. In *3rd International Workshop on Traffic Monitoring and Analysis (TMA)*, Vienna, Austria, Apr. 2011.

[30] H. Ringberg. *Privacy-Preserving Collaborative Anomaly Detection*. PhD thesis, Princeton University, 2009.

[31] G. Sathya Narayanan, T. Aishwarya, A. Agrawal, A. Patra, A. Choudhary, and C. Pandu Rangan. Multi party distributed private matching, set disjointness and cardinality of set intersection with information theoretic security. In J. Garay, A. Miyaji, and A. Otsuka, editors, *Cryptology and Network Security*, volume 5888 of *Lecture Notes in Computer Science*, pages 21–40. Springer Berlin / Heidelberg, 2009.

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

[33] Q. Ye, H. Wang, and C. Tartary. Privacy-preserving distributed set intersection. *Availability, Reliability and Security, International Conference on*, 0:1332–1339, 2008.

[34] V. Yegneswaran, P. Barford, and S. Jha. Global Intrusion Detection in the DOMINO Overlay System. In *Network and Distributed System Security Symposium (NDSS)*, 2004.