Achieving Full Security in Privacy-Preserving Data Mining

Marina Blanton
Department of Computer Science and Engineering
University of Notre Dame
mblanton@nd.edu

***Abstract*—In privacy-preserving data mining, a number of parties would like to jointly learn a function of their private data sets in a way that no information about their inputs, beyond the output itself, is revealed as a result of such computation. Yang et al. 2010 showed that several popular data mining algorithms can be reduced to three basic operations, secure implementation of which – termed Secure Product of Summations (SPoS), Secure Ratios of Summations (SRoS), and Secure Comparison of Summations (SCoS) – would lead to privacy-preserving data mining solutions. The authors showed that prior privacy-preserving data mining solutions are unsatisfactory in presence of participants' collusion and they gave new implementation of these operations that were designed to sustain the collusion. In this work, we show that unfortunately the protocols of Yang et al. leak a significant amount of private information and are not secure even if no collusion takes place. We then show how these operations can be securely and efficiently realized in the same and stronger security models, which leads to fully secure solutions for many data mining algorithms.**

I. Introduction

Privacy-preserving data mining has received a significant amount of attention in the research literature in the recent years. This is not surprising given a vast growth of the amount of collected information, including sensitive data, that we might desire to analyze. In privacy-preserving data mining, the data are distributed across multiple sites and are considered private information. The data owners would like to mine on their collective data, but in a way that no information about their private data sets is available to other participants (except what can be deduced from the output of the computation).

In this distributed setting, we can distinguish between horizontally partitioned data, vertically partitioned data, and their hybrid. In horizontally partitioned data, each data owner has complete information about a distinct set of entities. In vertically partitioned data, on the other hand, all parties hold information about the same set of entities, but each possesses information about distinct attributes.

A recent work by Yang et al. [24] provided an important observation that several popular data mining algorithms can be realized in a privacy-preserving way by relying on secure realizations of the following functions: (i) product of summations, (ii) ratio of summations, and (iii) comparison of summations. In all of the above cases, the sums are computed by adding private inputs of all of the participants, after which either the product, ratio (division), or comparison is applied to the aggregate values. Examples of data mining algorithms that can be securely realized using secure implementations of the three functions above include:

- Probability distribution of a data set distributed among a number of parties can be securely estimated by using a privacy-preserving solution that computes the ratio of summations [24]. In particular, if each party has a number of observations drawn from the same probability distribution, the parties can jointly estimate the distribution by dividing the number of data items with a particular value (summed across all parties) by the total number of collected items (also summed across all parties).
- Jha et al. [16] showed that secure computation of k-means clustering on horizontally partitioned data can be reduced to a secure computation of weighted average computation. This is the same as computing a ratio (or division) of sums, and therefore can be achieved by a secure ratio of summations protocol.
- Similarly, secure computation of Naïve Bayes classifier on horizontally partitioned data can be performed by securely computing the ratio of summations [23].
- In k-means clustering on vertically partitioned data, on the other hand, the closest cluster is determined by computing the index of the cluster with the minimum sum (where the sum is over the participants' private data) [22]. This requires secure computation of comparison of summations. More generally, when a number of elements need to be privately sorted, so that only the ordering of the elements is revealed, secure comparison can be used as well.

Therefore, secure computation of the summation (addition), product (multiplication), ratio (division), and comparison, where the private inputs are contributed by $n > 2$ parties, is the focus of this work.

Overview. Let $P_0, \ldots, P_{n-1}$ denote the $n > 2$ parties who contribute their data to the computation and learn the result. Yang et al. [24] show that in prior literature multi-party privacy-preserving protocols for the data mining algorithms mentioned above are susceptible to the problem of collusion. That is, solutions in [22], [18], [23] use special parties that are not expected to collude with any other participants (we refer the reader to [24] for additional information). Because this assumption can be very difficult to meet in practice, it is desirable to have solutions that are resistant to collusion by

subsets of the participants.

To address the problem, Yang et al. take an interesting approach in designing privacy-preserving protocols for product, ratio, and comparison of summations. Unlike computing over the integers, which is standard in secure multi-party computation, arithmetic is performed over real operands. All inputs of each party are assumed to be in the range $(0, 1)$, which means that they are scaled to that range before joint computation takes place. More precisely, the authors provide protocols for the following functionalities:

1) In *Secure Product of Summations (SPoS)* protocol, party P_i, for $i = 0, \ldots, n-1$, has input $x_1^{(i)}$ and $x_2^{(i)}$, where $x_j^{(i)} \in (0, 1)$ for $j = 1, 2$. As a result, each party learns $p = \sum_{i=0}^{n-1} x_1^{(i)} \cdot \sum_{i=0}^{n-1} x_2^{(i)}$.
2) In *Secure Ratios of Summations (k-SRoS)* protocol, party P_i, for $i = 0, \ldots, n-1$ has input $x_1^{(i)}, x_2^{(i)}, \ldots, x_k^{(i)}$, where $x_j^{(i)} \in (0, 1)$ for $j = 1, \ldots, k$. As a result, each party learns $r = \sum_{i=0}^{n-1} x_1^{(i)} : \sum_{i=0}^{n-1} x_2^{(i)} : \cdots : \sum_{i=0}^{n-1} x_k^{(i)}$.
3) In *Secure Comparison of Summations (SCoS)* protocol, party P_i, for $i = 0, \ldots, n-1$, has input $x_1^{(i)}$, and $x_2^{(i)}$, where $x_j^{(i)} \in (0, 1)$ for $j = 1, 2$. As a result, each party learns $l = \arg\max_{k=\{1,2\}} (\sum_{i=0}^{n-1} x_k^{(i)})$.

The Secure Product of Summations protocol SPoS protocol is realized in [24] using homomorphic encryption, and the authors provide a rigorous security proof against semi-honest participants (see below for a definition). The Secure Ratios k-SRoS and Comparison of Summation SCoS protocols are then built using the SPoS protocol and were left without security analysis. In this work we show that unfortunately the latter two protocols leak a significant amount of information about private inputs. Furthermore, the functionality realized by the SPoS protocol can be achieved (over integers or fixed point values) in a straightforward manner using a number of underlying secure computation techniques. Secure solutions for the comparison and ratio (or division) operations, however, are more complex with the state of the art secure multi-party computation (SMC) techniques and require careful design. Therefore, one of the goals of this work is to show how the above functions can be securely and efficiently realized, so that they could be used to build secure solutions for data mining algorithms.

As another security-related aspect of the above protocol formulation, notice that the operations are *not composable*. That is, the result of the computation in each of SPoS, k-SRoS, and SCoS is revealed to the participants in the clear and therefore the protocols cannot be automatically used as building blocks in more complex protocols. This, in particular, means that only data mining algorithms that can be realized using a single invocation of k-SRoS protocol or a number of SCoS protocols where the results of the comparisons are not private (i.e., part of the output) are supported. We argue that composability is desirable for widening the applicability of the solutions and will result in the ability to securely evaluate a much richer set of functions. For example, privacy-preserving computation on vertically partitioned data often involves multiple applications of the product operation (e.g., in Naïve Bayes and association rule mining). Also, in association rules mining, finding frequent itemsets with support exceeding certain threshold k% relies on addition, multiplication, and comparison operations, which cannot be realized by a single protocol above, but can be realized by their combination. Finally, when the index of the closest cluster is computed in k-means on vertically partitioned data, repeated application of SCoS would lead to information about the distance ordering of all of the clusters, which is undesirable and can be eliminated if the comparison protocol is composable.

To further widen the scope of data mining algorithms that can be securely realized using these techniques, we decompose the above protocols into four distinct operations: addition, multiplication, comparison, and division. We then provide secure and composable realizations for each of them. This means that arguments to each operation are distributed among the participants and the outputs are also communicated in a distributed way. This enables secure evaluation of any number of these operations in any order. Then in the beginning, each party will distribute its inputs among all of the participants, and once the desired function is computed, the parties will combine their outputs to learn the result of the computation. This setup has an additional advantage in that the set of parties who hold the inputs does not have to be the same as the set of parties who carry out the computation. Similarly, the set of parties who receive the output can be different from the set of parties supplying the inputs, carrying out the computation, or both. This has the flexibility that, for instance, for efficiency reasons the computation can be carried out by a selected group of input owners on behalf of all input providers or even be outsourced to a number of computational servers.

To summarize, the contributions of this work are: (i) we analyze k-SRoS and SCoS protocols of Yang et al. and show that they do not satisfy necessary security guarantees and (ii) we show how secure and composable solutions for addition, multiplication, comparison, and division operations can be realized to support a broader range of data mining algorithms. These solutions are secure in the same as in [24] and stronger security models.

Security model. In secure multi-party computation (SMC), there are two standard security models with respect to the malicious behavior of the participants. In the semi-honest (or honest-but-curious or passive) model, the participants follow the computation as prescribed, but might attempt to compute additional information from the messages observed during protocol execution. In the malicious (or active) model, the participants can arbitrarily deviate from the protocol, including aborting the computation, substituting wrong values for the intermediate results, etc. In both models, the participants can collude (i.e., share their information and coordinate their actions), which is modeled by an adversary that corrupts a number of participants. In the semi-honest model, security is achievable even if there is only a single honest participant. In

the malicious model, however, the number of honest parties is required to be a larger fraction of the participants (e.g., if all corrupted participants quit, the honest parties should be able to bring the computation to completion).

Publications that provide privacy-preserving solutions for data mining algorithms, including Yang et al., normally assume the semi-honest model (a number of publications assume even weaker security model, in which semi-honest participants do not collude). We therefore next formalize the security definition for that model.

Definition 1: Let $P_0, \ldots, P_{n-1}$ engage in protocol π that computes function $f(\mathsf{in}_0, \ldots, \mathsf{in}_{n-1}) = (\mathsf{out}_0, \mathsf{out}_1, \ldots, \mathsf{out}_{n-1})$, where in_i and out_i are the input and output of party P_i, respectively. 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 execution of protocol π. In particular, P_i's view is formed by its input, internal random coin tosses r_i, and messages $m_1, \ldots, m_s$ passed between the parties during protocol execution, i.e.,

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

The view of I is then 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 I of size at most t there exists a probabilistic polynomial time simulator S_I such that

$$\{S_I(\mathsf{in}_I, f(\mathsf{in}_0, \ldots, \mathsf{in}_{n-1}))\} \equiv \{\mathrm{VIEW}_\pi(I), \mathsf{out}_I\},$$

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 (using an appropriate security parameter).

Standard techniques for converting a solution secure in the semi-honest model to a solution secure in the malicious model ensure that each participant computes the next step of the computation correctly from the results of the previous steps. Such techniques are available in the literature for certain underlying secure computation mechanisms.

While the majority of publications on privacy-preserving data mining (including the work of Yang et al.) assume that the data owners will be conducting the secure collaborative computation themselves, this does not need to be the case. In particular, we distinguish between input parties (IP), computational parties (CP), and output parties (OP), which can be formed by distinct or overlapping sets of participants. In the current setting of data mining, it would be meaningful to have the parties who contribute their input to also receive the output, i.e., sets IP and OP to be the same. The computational parties can be chosen by the participants to minimize the possibility of collusion or other forms of misbehavior. For example, the participants can employ computational cloud providers and/or competing businesses to assume the role of computational servers. Then prior to the computation, each input provider distributes its input among the computational servers (in such a way that t or less colluding servers cannot recover the input), the servers carry out the secure computation, at the end of which each server communicates its share of the result to an output party. Upon receiving output information from the computational servers, each output parties reconstructs and learns the result of the computation.

The benefits of separating the input parties from computational parties include greater flexibility of the solution and lower computational cost under similar trust guarantees (e.g., when the number of computational parties is lower than the number of input providers, but each of them is more trusted than an average data provider).

II. Security analysis of techniques of Yang et al.

In this section we analyze two protocols – k-SRoS and SCoS – from [24]. In the subsequent analysis we assume that a fully secure realization of the SPoS protocol, on which k-SRoS and SCoS protocols are built, exists.

The Secure Ratios of Summations protocol is given in [24] as follows:

Protocol k-SRoS

Input: Each party P_i has input $x_1^{(i)}, x_2^{(i)} \ldots, x_k^{(i)}$, where each $x_j^{(i)} \in (0, 1)$.

Output: Each party learns $r = \sum_{i=0}^{n-1} x_1^{(i)} : \sum_{i=0}^{n-1} x_2^{(i)} : \cdots : \sum_{i=0}^{n-1} x_k^{(i)}$.

Protocol steps:

1) Each P_i generates a random real number $c^{(i)}$ in $(0, 1)$.
2) For $j = 1, \ldots, k$, the parties execute SPoS on private inputs $(x_j^{(0)}, c^{(0)}), \ldots, (x_j^{(n-1)}, c^{(n-1)})$ and learn $x_j = (x_j^{(0)} + \cdots + x_j^{(n-1)}) \cdot (c^{(0)} + \cdots + c^{(n-1)})$.
3) Each P_i computes $r = x_1 : x_2 : \cdots : x_k$.

We next show that this protocol leaks a significant amount of unintended information about the inputs $x_i^{(j)}$. This information cannot be deduced from the output of the protocol, which means that the protocol violates the security properties of Definition 1 which was also used in [24].

First, notice that the magnitude of the product x_j provide information about the magnitude of the sum $\sum_{i=1}^{n} x_j^{(i)}$, which cannot be inferred from the output ratio. In particular, while the value of the product x_j can lie in the range $(0, n^2)$, a specific value of x_j will make a part of that interval unreachable, thus revealing information about x_j. That is, the value of $\sum_{i=1}^{n} c^{(i)}$ is sampled from the distribution of the sum of n independent uniformly distributed variables from the range $(0, 1)$, which is known and fixed for a fixed value of n. This means that any party can rule out unlikely values for $\sum_{i=1}^{n} c^{(i)}$ and narrow the possible range for the sum $\sum_{i=1}^{n} x_j^{(i)}$ even further. We illustrate this analysis on an example.

Example 1. Let $n = 3$ and $k = 2$. Also let $x_1^{(0)} = 0.18$, $x_1^{(1)} = 0.23$, $x_1^{(2)} = 0.19$, $x_2^{(0)} = 0.19$, $x_2^{(1)} = 0.15$, $x_2^{(2)} = 0.20$, $c^{(0)} = 0.36$, $c^{(1)} = 0.89$, and $c^{(2)} = 0.18$. The only information that the parties obtain in a secure realization of this function is $r = \sum_{i=0}^{n-1} x_1^{(i)} : \sum_{i=0}^{n-1} x_2^{(i)} = 1 : 0.9$. From this information the parties can deduce that $\sum_{i=0}^{n-1} x_2^{(i)} < 2.7$ because $\sum_{i=0}^{n-1} x_1^{(i)}$ must be less than 3. If the granularity of

the computation is $\frac{1}{\ell}$, the parties also know that that $\sum_{i=0}^{n-1} x_1^{(i)}$ cannot take a few smallest possible values (such as $\frac{1}{\ell}$, $\frac{2}{\ell}$, etc.) since $\sum_{i=0}^{n-1} x_2^{(i)}$ is smaller and must satisfy the ratio r.

During the execution of the protocol above, on the other hand, the parties learn $x_1 = 0.858$ and $x_2 = 0.7722$. Because $\sum_{i=0}^{n-1} c^{(i)} < 3$, from the released information the parties immediately know that $\sum_{i=0}^{n-1} x_1^{(i)} > 0.858/3 = 0.286$ and $\sum_{i=0}^{n-1} x_2^{(i)} > 0.7722/3 = 0.2574$. This reduces the range of the possible values of $\sum_{i=0}^{n-1} x_1^{(i)}$ and $\sum_{i=0}^{n-1} x_2^{(i)}$ by about 10%. The parties might also be able to limit the value of $\sum_{i=0}^{n-1} x_1^{(i)}$ and $\sum_{i=0}^{n-1} x_2^{(i)}$ from the above when ℓ is small, i.e., because $\sum_{i=0}^{n-1} c^{(i)} \geq \frac{3}{\ell}$, it must hold that $\sum_{i=0}^{n-1} x_1^{(i)} \leq \frac{0.858\ell}{3}$ and $\sum_{i=0}^{n-1} x_2^{(i)} \leq \frac{0.7722\ell}{3}$.

We also know that the probability density function (pdf) for the sum of $n = 3$ uniform variables in (0, 1) is

$$f(x) = \begin{cases} \frac{1}{2}x^2 & 0 \leq x \leq 1 \\ \frac{1}{2}(-2x^2 + 6x - 3) & 1 \leq x \leq 2 \\ \frac{1}{2}(x^2 - 6x + 9) & 2 \leq x \leq 3 \end{cases}$$

with the mean 1.5 and variance 0.25. By integrating $f(x)$, we obtain the cumulative density function (cdf) $F(x)$. Let C denote a random variable from the distribution of which $\sum_{i=0}^{2} c^{(i)}$ is sampled. Then $F(s) = \Pr[C \leq s] = 0.1$ when $s \approx 0.8434$, which means that a value sampled from C will be at most 0.8434 with 10% probability. Similarly, $F(s) = \Pr[C \leq s] = 0.9$ when $s \approx 2.1566$, which gives us that with probability 80% any given value of $\sum_{i=0}^{2} c^{(i)}$ will lie in the interval $[0.8434, 2.1566]$. This means that the parties can discover that with 80% probability $\frac{0.858}{2.1566} \approx 0.398 \leq \sum_{i=1}^{n} x_1^{(i)} \leq \frac{0.858}{0.8434} \approx 1.017$, thus narrowing the range for the inputs even further. Similarly, with 66.67% probability any given value of $\sum_{i=0}^{2} c^{(i)}$ will lie in the range $[1, 2]$, which in this example implies that the parties learn that with the probability 66.67% $0.429 \leq \sum_{i=1}^{n} x_1^{(i)} \leq 0.858$. Using the same analysis, similar information can be learned about the range of the sum of $x_2^{(i)}$'s.

The above example showed what information is readily available to all participants. The situation, however, worsens when some of the participants collude. The protocols in [24] were intended to be resilient against any number of colluding parties, and we next show that the SRoS protocol is far from secure in that case. Once again, we need to consider what information the parties can deduce from the output of the (secure) computation and compare it to the information that the parties can discover during this protocol. Suppose that P_0 through P_{m-1} collude for some $1 < m < n$. Given the output of the computation $r = \sum_{i=0}^{n-1} x_1^{(i)} : \sum_{i=0}^{n-1} x_2^{(i)}$ (which will be only partial output when $k > 2$) and their respective inputs $\sum_{i=0}^{m-1} x_1^{(i)}$ and $\sum_{i=0}^{m-1} x_2^{(i)}$, the colluding parties can learn additional information from the output. In particular, they can find the relationship between the honest parties' inputs $\sum_{i=m}^{n-1} x_1^{(i)}$ and $\sum_{i=m}^{n-1} x_2^{(i)}$ by expressing the former as the function of the latter and somewhat narrow the known range for the possible values for $\sum_{i=m}^{n-1} x_1^{(i)}$ and $\sum_{i=m}^{n-1} x_2^{(i)}$ using the fact that both must lie in the range $(0, n - m)$.

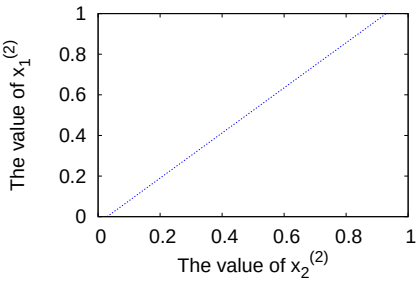

Fig. 1. Information about the input values of party P_2 that can be deduced from the output of a secure ratio protocol.

In the SRoS protocol of [24], however, much more information about private inputs of honest parties can be deduced. This is due to the fact that additional information about $c^{(i)}$'s (which are used to protect the inputs) is known and additional functions of the inputs of honest parties are revealed, which narrow their range. We illustrate the analysis on an example.

Example 2. For simplicity, we use the same values as in the previous example. Suppose that P_0 and P_1 collude against P_2. In a secure implementation, the colluding parties only know that $(0.41 + x_1^{(2)})/(0.34 + x_2^{(2)}) = 10/9$. They can compute the linear function $x_1^{(2)} = \frac{10}{9}x_2^{(2)} - 0.0322$, which is plotted in Figure 1. The parties then determine that in order for both $x_1^{(2)}$ and $x_2^{(2)}$ to lie in the range $(0, 1)$, the value of $x_2^{(2)}$ must be in the range $(0.029, 0.929)$. In other words, the range of possible values for $x_2^{(2)}$ is reduced by 8%.

In the protocol of [24], however, the parties know that $(0.41 + x_1^{(2)})(1.25 + c^{(2)}) = 0.858$ and $(0.34 + x_2^{(2)})(1.25 + c^{(2)}) = 0.7722$. Therefore, they can express $x_1^{(2)}$ and $x_2^{(2)}$ as linear functions of $c^{(2)}$ and, as before, attempt to limit the range of values for $x_1^{(2)}$ and $x_2^{(2)}$. In our example, these functions are $x_1^{(2)} = \frac{0.858}{1.25+c^{(2)}} - 0.41$ and $x_2^{(2)} = \frac{0.7722}{1.25+c^{(2)}} - 0.34$, which are plotted in Figure 2. It is obvious from the figure that the information leakage is significant and substantially exceeds what can be deduced from a secure implementation of the computation. In particular, the range of $x_1^{(2)}$ is narrowed down to $[0, 0.2764]$ and the range of $c^{(2)}$ is narrowed down to $[0, 0.8427]$ using the first function. Using the second function, the range of $x_2^{(2)}$ is narrowed down to $[0.0032, 0.2778]$, but can further be reduced to $[0.0295, 0.2778]$ considering that the value of $c^{(2)}$ can be at most 0.8427 (this value of $c^{(2)}$ is shown in the plot for $x_2^{(2)}$ in Figure 2). Since the range of possible values is reduced by 72.4% for $x_1^{(2)}$ and by 76.2% for $x_2^{(2)}$, it is clear that a large amount of unintended information is leaked.

We now analyze the second protocol, Secure Comparison of Summations, which is given in [24] as follows:

Protocol SCoS

Input: Each party P_i has private input $x_1^{(i)}, x_2^{(i)} \ldots$, where each $x_j^{(i)} \in (0, 1)$. There is also public integer $P > 1$.

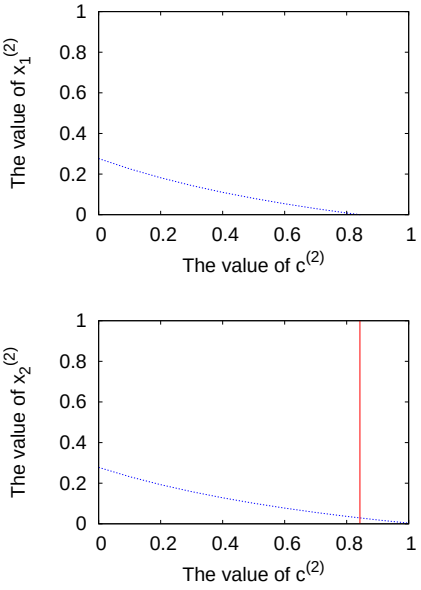

Fig. 2. Information about the input values of party P_2 that can be deduced from the execution of k-SRoS protocol.

Output: Each party learns a bit b, where $b = 0$ if $\sum_{i=0}^{n-1} x_1^{(i)} > \sum_{i=0}^{n-1} x_2^{(i)}$ and $b = 1$ otherwise.

Protocol steps:

1) Each P_i generates a random real $u^{(i)}$ in $(0, 1-1/P)$ and computes $y_1^{(i)} = x_1^{(i)}/P + u^{(i)}$ and $y_2^{(i)} = x_2^{(i)}/P + u^{(i)}$.
2) Each P_i generates a random real $c^{(i)} \in (0, 1)$, and the parties execute SPoS twice:
   (i) on private inputs $(y_1^{(0)}, c^{(0)}), \ldots, (y_1^{(n-1)}, c^{(n-1)})$ to learn $p_1 = (y_1^{(0)} + \cdots + y_1^{(n-1)})(c^{(0)} + \cdots + c^{(n-1)})$,
   (ii) on private inputs $(y_2^{(0)}, c^{(0)}), \ldots, (y_2^{(n-1)}, c^{(n-1)})$ to learn $p_2 = (y_2^{(0)} + \cdots + y_2^{(n-1)})(c^{(0)} + \cdots + c^{(n-1)})$.
3) Each P_i outputs 0 if $p_1 > p_2$, and 1 otherwise.

For conciseness, we only analyze the case when a collusion of participants takes place. As before, in a secure implementation, the participants can only learn information that can be deduced from their inputs and the output bit. If parties P_0 through P_{m-1} collude, the only information they learn is the constraints on the sums $\sum_{i=m}^{n-1} x_1^{(i)}$ and $\sum_{i=m}^{n-1} x_2^{(i)}$ that lead to the computed bit.

In the comparison protocol of [24], however, additionally information about the magnitude of the difference $\sum_{i=0}^{n-1} x_1^{(i)} - \sum_{i=0}^{n-1} x_2^{(i)}$ can be deduced even without collusion. When collusion is present, this can additionally lead to information about the inputs of a particular participant. We next analyze the protocol in more detail.

While no recommended value for P was given in [24], when $P = 2^\kappa$ for a security parameter $\kappa > 80$, the release of a single p_j statistically hides the value of $\sum_{i=m}^{n-1} x_j^{(i)}$ even in presence of collusion. However, because the value of each $u^{(i)}$ is reused in both p_1 and p_2, security is violated. That is, by computing

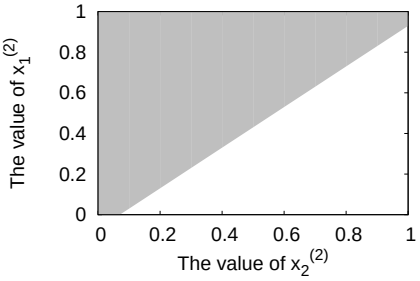

Fig. 3. Information about the input values of P_2 that can be deduced from the output of a secure comparison protocol.

$(p_1 - p_2)P = \left(\sum_{i=0}^{n-1} x_1^{(i)} - \sum_{i=0}^{n-1} x_2^{(i)}\right) \sum_{i=1}^{n} c^{(i)}$ information about the magnitude of the difference between inputs $x_1^{(i)}$ and $x_2^{(i)}$ is gained. When some parties collude, this can lead to learning information about the inputs of a single individual.

Example 3. Let us use the same set of values for n, $x_1^{(i)}$, $x_2^{(i)}$, and $c^{(i)}$ as in the previous examples. That is, $n = 3$, $x_1^{(1)} = 0.18$, $x_1^{(2)} = 0.23$, $x_1^{(3)} = 0.19$, $x_2^{(1)} = 0.19$, $x_2^{(2)} = 0.15$, $x_2^{(3)} = 0.20$, $c^{(1)} = 0.36$, $c^{(2)} = 0.89$, and $c^{(3)} = 0.18$. In secure implementation and in presence of a collusion between parties P_0 and P_1, they know from the output that $(0.41 + x_1^{(2)}) > (0.34 - x_2^{(2)})$. This allows the parties to slightly restrict the inputs of P_2, where now $x_1^{(2)} \in (0.07, 1)$, $x_2^{(2)} \in (0, 0.93)$, and the possible choices for the pair $(x_1^{(2)}, x_2^{(2)})$ are given in the shaded area in Figure 3.

After executing the protocol, however, the colluding parties P_0 and P_1 learn that $(p_1 - p_2)P = (0.07 + (x_1^{(2)} - x_2^{(2)}))(1.25 + c^{(2)}) = 0.0858$. This allows them to compute the value of $(x_1^{(2)} - x_2^{(2)})$ as a function of $c^{(2)}$, which is shown in the top plot of Figure 4. That is, the value of $(x_1^{(2)} - x_2^{(2)})$ is restricted to the range $(-0.0014, -0.0319)$. The parties can thus conclude that $x_2^{(2)} - 0.0319 < x_1^{(2)} < x_2^{(2)} - 0.0014$. By combining this information with what is already known from the output alone (Figure 3), the parties can limit the possible values of pairs $(x_1^{(2)}, x_2^{(2)})$ to the area shaded in the bottom plot of Figure 4. We can see that the area shaded in Figure 3, which corresponds to uncertainty about P_2's inputs in secure execution, is reduced by 94.7% to the area shaded in Figure 4 as a result of SCoS execution.

III. Secure Computation of Data Mining Functions

This section shows how the operations identified in Section I as the basis of many data mining protocols – namely, addition, multiplication, comparison, and division – can be implemented in a secure and composable way. Composability means that, due to the theorem of Canetti [6], a protocol consisting of secure sub-protocols will be itself secure.

The choice of the techniques used in this work was driven by the practical SMC efficiency considerations. In particular, SMC can be based on three general types of techniques: (i) threshold homomorphic encryption, (ii) garbled circuit

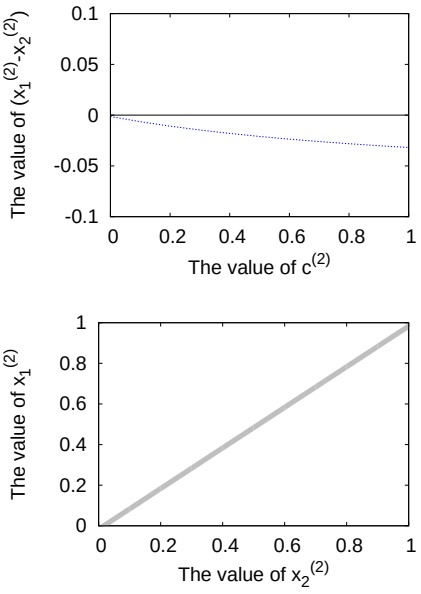

Fig. 4. Information about the input values of P_2 that can be deduced from the execution of SCoS protocol.

evaluation, and (iii) secret sharing. The disadvantage of homomorphic encryption is that all operations take place in a group of large size, which means that every single operation is expensive. Multi-party garbled circuit evaluation (GCE) (see, e.g., [14]) is generally more efficient per operation than homomorphic encryption-based techniques, but uses Boolean circuits that operate on bits. Furthermore, the most efficient implementation of multi-party GCE known today, FairplayMP [3], uses secret sharing techniques for building a garbled circuit, and several interactive operations over secret-shared values are needed for each gate. This leaves us with the techniques based on a linear secret sharing scheme, where all operations are performed over a field of small size and the computation of any linear combination of secret-shared values requires no interaction of the participating parties.

In detail, we assume that *computational* parties (CP) $P_0, \ldots, P_{n-1}$ are connected with each other by secure authenticated channels. Each input party also establishes secure channels with P_0 through P_{n-1}. For concreteness, we assume that Shamir secret sharing [21] over a finite field $\mathbb{Z}_p$ is used as the underlying secret sharing scheme, where p is a (small) prime (such that all values we want to represent do not exceed p). We use notation $[a]_p$ to denote that a is shared among the parties. In particular, when value s is shared, there is a polynomial $f_s(x) = a_t x^t + \cdots + a_1 x + s \bmod p$ with randomly chosen coefficients $a_i \in \mathbb{Z}_p$ for $i = 1, \ldots, t$ and P_i's share is $f_s(i+1)$. With this setup, the shared value s can be easily reconstructed from any $t+1$ shares using Lagrange interpolation. On the other hand, t parties or less cannot learn *any information* about the shared value, i.e., it is perfectly protected in information-theoretic sense regardless of the computational power of the participants. Therefore, an adversary can corrupt up to t parties, where $t < n/2$ for passive adversaries and additional restrictions can apply for active adversaries.

Let values a and b be shared among the parties as $[a]_p = (f_a(1), \ldots, f_a(n))$ and $[b]_p = (f_b(1), \ldots, f_b(n))$. Then the following operations can be performed on the shared values without any interaction: to compute $[a + b \bmod p]_p$, each party P_i locally computes $f_a(i) + f_b(i) \bmod p$; to compute $[c + a \bmod p]_p = c + [a]_p$ where c is a publicly known value, each party P_i locally computes $c + f_a(i) \bmod p$; to compute $[ca \bmod p]_p = c[a]_p$, each party P_i computes $cf_a(i) \bmod p$. Similarly, any linear combination of any number of shared values can be computed locally.

Multiplication of shared values, on the other hand, $[ab \bmod p]_p = [ab]_p$ requires $P_0, \ldots, P_{n-1}$, to communicate with each other. In particular, the overhead is dominated by the need for party P_i to securely transmit an ℓ-bit message to each party (and receive a message from each party), e.g., using the multiplication protocol from [13]. Therefore, communication overhead for a single party is $n-1$ transmissions, and a multiplication protocol involves transmission of the total of $O(n^2)$ messages. The current description assumes that the adversary is passive. When security against malicious adversaries is desired, additional general techniques based on verifiable secret-sharing (VSS) can be utilized to make the protocols resilient to such behavior (see, e.g., [13], [10]).

Normally, performance of a protocol is measured in terms of two parameters: (i) the number of interactions (multiplications, secret splitting or opening) necessary to perform the computation and (ii) the number of sequential interactions, i.e., rounds. We employ the same metric in this work.

Going back to our setup in Section I, before a privacy-preserving data mining computation takes place, each IP distributes its private inputs to the CPs $P_0, \ldots, P_{n-1}$. Upon protocol completion, $P_0, \ldots, P_{n-1}$ communicate their shares of the output to each OP who reconstructs the result from the shares. Two secure operations – addition and multiplication – are trivial in this framework: addition of shares is performed by each CP locally and multiplication is performed interactively as described above. The rest of this section concentrates on the remaining two operations: comparison and division.

A. Comparison

Performing comparison of two values often requires access to the bits in their binary representation, and we use notation $[a]_B = [a_1]_p [a_2]_p \ldots [a_\ell]_p$ to denote that each bit of binary representation of $a = \sum_{i=1}^{\ell} 2^i a_i$ is secret-shared by the parties. Because comparison is a commonly used operation, it has been a subject of research. Some publications provide techniques for comparing two bit-decomposed values, the most efficient of which are listed in Table I, while others such as [7] allow comparison without bitwise representation. We defer discussion of the latter to section III-D. Theoretical literature (such as [11]) concentrates on constant-round techniques, which normally minimize the latency of computation when the operation is run in isolation. When, on the other hand, many

Source	Rounds	Interactions	Security	Passive	Active
[11]	8	19ℓ	perfect	$\surd$	$\surd$
[12]	$\log_2 \ell$	$3\ell - 2$	perfect	$\surd$	$\surd$

TABLE I
KNOWN TECHNIQUES FOR COMPARISON OF TWO BIT-DECOMPOSED VALUES OF LENGTH ℓ.

Source	Rounds	Interactions	Security	Pas	Act
[11]	15	$47\ell \log_2 \ell$	perfect	$\surd$	$\surd$
[19]	5	$7\ell + 3$	statistical	$\surd$	$\surd$
[7], [1]	$\log_2 \ell + 1$	$\ell \log_2 \ell + \ell$	perfect	$\surd$	$\surd$
[7], [1]	$2 \log_2 \ell - 1$	$5\ell - 2 \log_2 \ell - 4$	perfect	$\surd$	$\surd$
Appendix A	ℓ	$2\ell - 1$	perfect	$\surd$	$\surd$

TABLE II
KNOWN TECHNIQUES FOR ADDITION OF TWO BIT-DECOMPOSED VALUES OF LENGTH ℓ.

operations can be executed in parallel, techniques that minimize the overall work are preferred. For example, in Table I the techniques of [12] offer lower round and overall overhead as long as $\ell \leq 256$, i.e., for all data mining applications. The last three columns in the table show that perfect information-theoretic security can be achieved in presence of passive or active adversaries.

B. Bit-oriented addition and subtraction

Before discussing the division operation, we turn our attention to addition and subtraction of bit-decomposed values $[a]_B$ and $[b]_B$. While addition of two secret-shared values $[a]_p$ and $[b]_p$ can be performed without any interaction, in some cases it is beneficial to be able to add two values in a bit-decomposed form, so that their sum is also available in the bit-decomposed form (the same functionality can be achieved by locally computing $[a]_p$ and $[b]_p$ from $[a]_B$ and $[b]_B$, respectively, as $[x]_p = \sum_{i=1}^{\ell} 2^i [x_i]_p$, then locally adding them $[c]_p = [a+b]_p$ and decomposing $[c]_p$ into bits, but adding two bitwise values can often be achieved more efficiently than bit decomposition). In particular, this operation is used in division and bit decomposition protocols. We list known techniques for bit-oriented addition in Table II (subtraction can be performed similarly). Also, while prior publications target minimizing the number of rounds, the best performance in terms of the overall overhead is achieved using a linear (in the length of values) number of rounds. For that reason, we design addition and subtraction protocols that use only $2\ell - 1$ multiplications. They are described in Appendix A.

In Table II, statistical security means that with negligible probability (in security parameter κ) some information about private values can be revealed. Protocols with statistical security often have to use a field of a larger size resulting in slower operations.

C. Division

The division operation is most complex among the computational primitives considered in this work. Privacy-preserving division protocols appeared in the literature starting from two-party solutions based on homomorphic encryption [2], [4] to more recent multi-party protocols [8], [15], [9]. There are a number of (conventional) division algorithms that can be used as the basis for secure implementation such as long division, Newton-Raphson, etc. Perhaps the most efficient privacy-preserving division protocol is that of Catrina and Saxena [9] that uses Goldschmidth's method and was designed to work on fixed-point values. While that algorithm has faster convergence (i.e., requires fewer iterations) than long division which computes a single bit of the quotient per iteration, the protocol of [9] is not guaranteed to compute the exact result (without an error). Furthermore, the solution achieves statistical security and has to rely on fields of large size, and security against an active adversary was not shown. We therefore provide an alternative solution that computes the result exactly, uses only standard techniques for which countermeasures against active adversaries are known, and can have advantageous performance when the length of values is small.

The solution is built using an intuitive implementation of division, which consists of a sequence of comparisons and subtractions. That is, the logic of the protocol for computing $q = \lfloor v/d \rfloor$ and $r = v \bmod d$ is as shown below, where r holds the current remainder:

1. $r := v$;
2. for $i = 1, \ldots, \ell$
3. $\quad q_{\ell-i} := (r \stackrel{?}{\geq} 2^{\ell-i} d)$;
4. $\quad r := r - q_{\ell-i} 2^{\ell-i} d$;
5. output $q_1, \ldots, q_\ell$;

In [15] we built a secure division protocol using this logic and homomorphic encryption, but homomorphic encryption makes it too expensive for use in applications that should scale. Here we start with an optimization suggested in [15], then describe our own additional optimizations, and use more efficient building blocks to develop a protocol with fast performance.

Because on line 3 we perform a comparison with $2^{\ell-i} d$, a straightforward implementation would involve computation on values as large as $2\ell - 1$ bits. It is, however, noted in [15] that $(\ell + 1)$-bit representation is sufficient for all iterations of the computation. In particular, the $(2\ell - i)$-bit representation of $2^{\ell-i} d$ is formed by shifting the bits of the bit-decomposed d $\ell - i$ positions to the left and appending $\ell - i$ bits corresponding to zeros. The $(\ell+1)$-bit representation of $2^{\ell-i} d$ can be formed by also appending $\ell - i$ zero bits to the value of d, but replacing the $\ell - i$ most significant bits of the resulting representation with a single bit that corresponds to their OR. This means that if at least one of the $\ell - i$ most significant bits of $2^{\ell-i} d$ is 1, the relationship $v < 2^{\ell-i} d$ will be preserved (here the $(\ell + 1)$-bit representation of v is formed by prepending a zero as its most significant bit). If, however, all of such bits are 0, the value of $(\ell + 1)$-bit representation of $2^{\ell-i} d$ equals to the value of its $(2\ell - i)$-bit representation. Notice that the $(\ell + 1)$st bits for each $2^i d$ can be computed once for all i before entering the loop as prefix-OR operation $t_i = \bigvee_{j=0}^{i-1} d_{\ell-j}$ for $i = 1, \ldots, \ell$.

To further optimize the computation, we consider reducing the number of iterations in the division. While the number of iterations is not reduced asymptotically, our optimizations reduce their number by a constant multiplicative factor with minimal impact on the communication and computation of the protocol. The optimization consists of computing a fixed number k of quotient bits within a single iteration of the protocol. The idea consists of replacing the comparison on line 3 with $2^k - 1$ parallel comparisons that will compare v with values $2^{\ell-i-k+1}d \cdot j$, where j ranges from 1 to $2^k - 1$. Then k bits of q can be determined from the outcomes of the comparisons in $k-1$ rounds using $\frac{1}{2}(k-1)k$ multiplications. Let $c_{i,j} := (r \stackrel{?}{\geq} 2^{\ell-i-k+1}d \cdot j)$ denote the outcome of a comparison. We set $q_{\ell-i} = c_{i,2^k-2^{k-1}} = c_{i,2^{k-1}}$, $q_{\ell-i-1}$ is computed as $q_{\ell-i-1} = q_{\ell-i}c_{i,2^k-2^{k-2}} + (1-q_{\ell-i})c_{i,2^{k-1}-2^{k-2}}$, etc. Finally, the value of $2^{\ell-i-k+1}d \cdot j$ is subtracted from r, where $j = \prod_{t=0}^{k-1} 2^{k-1-t}q_{\ell-i-t}$.

To accomplish the last step, subtraction must be performed without having access to the computed quotient bits. We achieve this by forming the value to be subtracted as $\sum_{j=1}^{2^k-1} i_j(2^{\ell-i-k}d \cdot j)$, where i_j is a bit and at most one i_j is set to 1. The value of i_j is computed from the quotient bits $q_{\ell-i}, \ldots, q_{\ell-i-k+1}$. For example, when $k = 2$, we obtain $i_1 = \neg q_{\ell-i} \wedge q_{\ell-i-1}$, $i_2 = q_{\ell-i} \wedge \neg q_{\ell-i-1}$, and $i_3 = q_{\ell-i} \wedge q_{\ell-i-1}$. If the values $d \cdot j$ are precomputed in bitwise form in the beginning of the protocol, computation of $i_j(d \cdot j)$ consists of multiplying each bit of $d \cdot j$ with i_j, after which the result is appended with $\ell - i - k + 1$ zero bits to obtain $i_j(2^{\ell-i-k+1}d \cdot j)$. Finally, the sum across all j's is performed (on bit-decomposed values) locally because at most one value being added is non-zero.

The above optimization technique for reducing the number of rounds affects the way the comparisons are carried out on line 3 above using $(\ell+1)$-bit values, and we need to ensure that $(\ell+1)$-bit representations are computed for all $2^{\ell-i-k+1}d \cdot j$. In order to combine both optimization techniques on line 4, we notice that it is sufficient to use only ℓ least significant bits of $2^{\ell-i-k+1}d \cdot j$ because the subtraction is performed only if the ℓ-bit value r is larger than $2^{\ell-i-k+1}d \cdot j$. This gives us the overall division protocol presented next.

For simplicity of exposition, we describe it for the case of $k = 2$ and even ℓ (when ℓ is odd, one iteration of the loop should be executed to compute a single bit of the quotient instead of two).

$\mathsf{Divide}([v]_B = [v_1]\ldots[v_\ell], [d]_B = [d_1]\ldots[d_\ell])$

1) The parties execute $[s]_B = \mathsf{Add}(0[d_1]\ldots[d_\ell], [d_1]\ldots[d_\ell]0)$, where $[s]_B = [s_1]\ldots[s_{\ell+2}]$.
2) The parties execute $([t_1], \ldots, [t_{\ell-1}]) = \mathsf{PrefixOR}([d_\ell], \ldots, [d_2])$ and $([t'_1], \ldots, [t'_\ell]) = \mathsf{PrefixOR}([s_{\ell+2}], \ldots, [s_3])$, also set $[t_0] = 0$.
3) Each party sets $[r]_B = [v]_B$, $[0d]_B = 0_B$, $[1d]_B = [d]_B$, $[2d]_B = 0[d_1]\ldots[d_{\ell-1}]$, and $[3d]_B = [s_1]\ldots[s_\ell]$.
4) For $i = 1, 3, \ldots, \ell - 1$
   a) the parties execute $[c_1] = \mathsf{Compare}([r]_B, 0^{\ell-i-1}[d_1]\ldots[d_{i+1}][t_{\ell-i-1}])$, $[c_2] = \mathsf{Compare}([r]_B, 0^{\ell-i}[d_1]\ldots[d_i][t_{\ell-i}])$, and $[c_3] = \mathsf{Compare}([r]_B, 0^{\ell-i-1}[s_1]\ldots[s_{i+1}][t'_{\ell-i+1}])$ in parallel, where 0^j denotes concatenation of j zeros.
   b) the parties set $[q_{\ell-i}] = [c_2]$ and compute $[q_{\ell-i-1}] = [q_{\ell-i}]([c_3] - [c_1]) + [c_1]$ using one multiplication.
   c) the parties compute $[q_{\ell-i}][q_{\ell-i-1}]$ and set $[i_1] = [q_{\ell-i-1}] - [q_{\ell-i}][q_{\ell-i-1}]$; $[i_2] = [q_{\ell-i}] - [q_{\ell-i}][q_{\ell-i-1}]$; and $[i_3] = [q_{\ell-i}][q_{\ell-i-1}]$.
   d) the parties compute in parallel $[o_{1,j}] = [i_1][1d_j]$, $[o_{2,j}] = [i_2][2d_j]$, and $[o_{3,j}] = [i_3][3d_j]$ for $j = 1, \ldots, i + k - 1$.
   e) each party locally sets $[o_j] = 0$ for $j = 1, \ldots, \ell - i - k + 1$ and $[o_{j+\ell-i-k+1}] = [o_{1,j}] + [o_{2,j}] + [o_{3,j}]$ for $j = 1, \ldots, i + k - 1$ to obtain $[o]_B = [o_1]\ldots[o_\ell]$.
   f) the parties execute $[r]_B = \mathsf{Subtract}([r]_B, [o]_B)$.
5) Return $[q]_B = [q_1]\ldots[q_\ell]$.

In the above, step 1 computes $3d$ (the values $0d$, $1d$, $2d$, and $3d$ are explicitly set in step 3), and step 2 computes prefix OR to form $(\ell+1)$-bit representation of $2^{\ell-i-1}d$, $2^{\ell-i-1}2d$, and $2^{\ell-i-1}3d$. In step 4(a), three comparisons with r are performed in parallel, after which 2 quotient bits are computed in step 4(b). Steps 4(c)–(e) compute the sum $\sum_{j=1}^{3} i_j(2^{\ell-i-1}d \cdot j)$, after which it is subtracted from r. Note that when the remainder $r = v \bmod d$ is not needed, steps 4(c)–(f) can be skipped in the last iteration of the loop.

Complexity. The protocol performs a single addition in step 1 and two prefix OR operations in step 2. The loop in step 4 adds 3 comparison operations, one subtraction, and a number of multiplications in $\frac{\ell}{2}$ iterations. Notice that similar complexity would be achieved if we performed ℓ iterations with a single comparison and subtraction and a number of multiplications, while the round complexity would rise by about a factor of 2. Assuming that we use bitwise addition and comparison operations with $\log_2 \ell$ rounds and PrefixOR with $\log_2 \ell$ rounds and $0.5\ell \log_2 \ell$ interactions from [7] with perfect privacy, we obtain the round complexity of $2\log_2 \ell + 1 + \frac{\ell}{2}(2\log_2 \ell + 4)$. For $\ell = 32$, this gives us 235 rounds. The number of multiplications is $1.5\ell \log_2 \ell + \ell + \frac{\ell}{2}(10\ell + \ell \log_2 \ell - 4) + 3\ell(\frac{\ell}{2} + 1)$. When several division or other operations are executed in parallel, the cost of each of them can be noticeably reduced at the expense of increasing the number of rounds. Also, with statistical security guarantees, both the number of rounds and the overall cost can be reduced by using more efficient PrefixOR and comparison operations from [7].

Security. As can be seen from the protocol, no single value is revealed throughout the entire computation, i.e., all values are handled in a secret shared form. This means that no information can be leaked except what the building blocks might leak. Because we use only secure sub-protocols to perform privacy-preserving division, by the composition theorem [6] it means that the entire protocol is also secure. In particular, the simulator in Definition 1 can perfectly simulate the view of the parties by using random values (or using the simulators

Source	Rounds	Multiplications	Security	Pas	Act
[11]	38	$94\ell \log_2 \ell + 93\ell$	perfect	$\surd$	$\surd$
[17]	25	$47\ell \log_2 \ell + 63\ell + 30\sqrt{\ell}$	perfect	$\surd$	$\surd$
[20]	12	$39.5\ell + 15$	statistical	$\surd$	$\surd$
[9], [1]	$\log_2 \ell + 2$	$\ell \log_2 \ell + \ell + 1$	statistical	$\surd$	?

TABLE III
KNOWN TECHNIQUES FOR BIT DECOMPOSITION OF AN ℓ-BIT VALUE.

for the building blocks) assuming that no more than t parties collude. Security against active adversaries can be achieved by using standard verifiable secret sharing techniques.

D. Bit decomposition and related operations

Performance of known techniques for securely computing ℓ least significant bits $[a_1]\ldots[a_\ell]$ from $[a]_p$ are summarized in Table III. Security of the solution in [9], [1] was not shown to hold in presence of active adversaries. We also note that the literature provides techniques for performing comparison without having to bit-decompose the operands first, and the performance is more efficient than applying bit decomposition followed by comparison of bit-decomposed values. Such techniques were given in [17], [5], [9] for comparison, equality, and range checks.

E. Performance

Finally, we show that the techniques described in this work are practical for many applications. Based on our experiments, a single round of multiplication takes about 3msec on a LAN with $n = 5$ parties, where a noticeable portion comes from the need to encrypt/decrypt communication. This means that addition, multiplication, and comparison operations are extremely fast, while a 235-round division algorithm can be executed in less than a second. This performance compares favorably with the performance results for this operation reported by Yang et al. [24] which do not tolerate collusion.

IV. CONCLUSIONS

This work follows the line of research taken by Yang et al. [24] that suggest that a large number of data mining problems can be realized in a privacy-preserving setting by designing techniques for what we decompose into secure evaluation of addition, multiplication, comparison, and division. We first show that, despite security claims given in [24], their protocols are not secure (i.e., leak a significant amount of private information) in presence of semi-honest adversaries even if no collusion between the participants takes place. We then show how efficient solutions secure in both semi-honest and malicious models can be developed in this framework. An additional advantage of our approach is that the protocols do not need to be run by the data owners themselves, but instead can be executed by a selected groups of parties or even outsourced to external computational servers.

ACKNOWLEDGMENTS

This work was supported in part by the grant AFOSR-FA9550-09-1-0223 from the Air Force Office of Scientific Research. Any opinions, findings, and conclusions or recommendations expressed in this publication are those of the author and do not necessarily reflect the views of the AFOSR.

REFERENCES

[1] SecureSCM Project Deliverable D9.2. http://pi1.informatik.uni-mannheim.de/index.php?pagecontent=site/Research.menu/SecureSCM.page, University of Mannheim, July 2009.
[2] M. Atallah, M. Bykova, J. Li, K. Frikken, and M. Topkara. Private collaborative forecasting and benchmarking. In *ACM Workshop on Privacy in the Electronic Society (WPES)*, pages 103–114, 2004.
[3] 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.
[4] P. Bunn and R. Ostrovsky. Secure two-party k-means clustering. In *ACM Conference on Computer and Communications Security (CCS)*, pages 486–497, 2007.
[5] M. Burkhart, M. Strasser, D. Many, and X. Dimitropoulos. SEPIA: Privacy-preserving aggregation of multi-domain network events and statistics. In *USENIX Security Symposium*, pages 223–240, 2010.
[6] R. Canetti. Security and composition of multiparty cryptographic protocols. *Journal of Cryptology*, 13(1):143–202, 2000.
[7] O. Catrina and S. de Hoogh. Improved primitives for secure multiparty integer computation. In *Security and Cryptography for Networks (SCN)*, pages 182–199, 2010.
[8] O. Catrina and C. Dragulin. Multiparty computation of fixed-point multiplication and reciprocal. In *International Workshop on Database and Expert Systems Application (DEXA)*, pages 107–111, 2009.
[9] O. Catrina and A. Saxena. Secure computation with fixed-point numbers. In *Financial Cryptography and Data Security (FC)*, pages 35–50, 2010.
[10] R. Cramer, I. Damgård, and U. Maurer. General secure multi-party computation from any linear secret-sharing scheme. In *Advances in Cryptology – EUROCRYPT*, volume 1807 of *LNCS*, pages 316–334, 2000.
[11] 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)*, pages 285–304, 2006.
[12] J. Garay, B. Shoenmakers, and J. Villegas. Practical and secure solutions for integer comparison. In *Public Key Cryptography (PKC)*, pages 330–342, 2007.
[13] 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.
[14] 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.
[15] T. R. Hoens, M. Blanton, and N. Chawla. A private and reliable recommendation system using a social network. In *IEEE International Conference on Information Privacy, Security, Risk and Trust (PASSAT)*, pages 816–825, 2010.
[16] S. Jha, L. Kruger, and P. McDaniel. Privacy-preserving clustering. In *European Symposium On Research In Computer Security (ESORICS)*, pages 397–417, 2005.
[17] T. Nishide and K. Ohta. Multiparty computation for interval, equality, and comparison without bit decomposition protocol. In *Conference on Theory and Practice of Public Key Cryptography (PKC)*, pages 343–360, 2007.
[18] M. Ozarar and A. Ozgit. Secure multiparty overall mean computation via oblivious polynomial evaluation. In *International Conference on Security of Information and Networks (SIN)*, pages 84–95, 2007.
[19] T. Reistad. Multiparty comparison – An improved multiparty protocol for comparison of secret-shared values. In *International Conference on Security and Cryptography (SECRYPT)*, pages 325–330, 2009.
[20] T. Reistad and T. Toft. Linear, constant-rounds bit-decomposition. In *International Conference on Information, Security and Cryptology (ICISC)*, pages 245–257, 2009.

[21] A. Shamir. How to share a secret. *Communications of the ACM*, 22(11):612–613, 1979.
[22] J. Vaidya and C. Clifton. Privacy-preserving k-means clustering over vertically partitioned data. In *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 206–215, 2003.
[23] J. Vaidya, M. Kantarcioglu, and C. Clifton. Privacy-preserving Naive Bayes classification. *VLDB Journal*, 17(4):879–898, 2008.
[24] B. Yang, H. Nakagawa, I. Sato, and J. Sakuma. Collusion-resistant privacy-preserving data mining. In *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 483–492, 2010.

APPENDIX A
SUPPLEMENTAL PROTOCOLS

Here we describe addition and subtraction algorithms that take ℓ-bit operands in bit-decomposed form and produce their $(\ell+1)$-bit sum or difference, respectively. The protocols involve ℓ rounds, but require only $2\ell-1$ interactive operations (multiplications), giving a performance advantage compared to other available solutions when many protocols are run in parallel.

A. Bit-oriented addition

In what follows, let c_i for $i = 1,\ldots,\ell+1$ denote the ith carry bit with $c_1 = 0$, and s_i for $i = 1,\ldots,\ell$ denote the ith bit of the sum. Then using the truth table for binary addition we can express s_i and c_{i+1} as a function of a_i, b_i, and c_i using Boolean operators as follows:

$$s_i = a_i \oplus b_i \oplus c_i \quad \text{and} \quad c_{i+1} = \neg c_i \wedge a_i \wedge b_i + c_i \wedge \neg(a_i \vee b_i).$$

We further rewrite c_{i+1} as an algebraic expression as:

$$\begin{aligned} c_{i+1} &= (1-c_i)a_ib_i + c_i(a_i + b_i - a_ib_i) \\ &= a_ib_i + c_i(a_i + b_i - 2a_ib_i). \end{aligned}$$

Because $a_i \oplus b_i = a_i + b_i - 2a_ib_i$, we see that all s_i's and c_{i+1}'s can be computed using only 2ℓ multiplications (more precisely, $2\ell-1$ multiplications because c_1 is known to be 0): ℓ multiplications to compute all a_ib_i and ℓ multiplications to compute all $c_i(a_i \oplus b_i)$. This computation results in ℓ rounds because of the sequential nature of the computation of the c_{i+1}'s. More precisely, we obtain the protocol given next. For simplicity of exposition, all products a_ib_i are computed in the first round of the protocol, while in an implementation it would make sense to spread them out through all of the rounds.

$\mathsf{Add}([a]_B = [a_1]\ldots[a_\ell], [b]_B = [b_1]\ldots[b_\ell])$

1) For $i = 1,\ldots,\ell$ in parallel execute $[d_i] = [a_i][b_i]$.
2) Each party locally sets $[f_i] = [a_i] + [b_i] - 2[d_i]$ for $i = 1,\ldots,\ell$ and also $[s_1] = [f_1]$ and $[c_2] = [d_1]$.
3) For $i = 2,\ldots,\ell$,
   a) execute $[h_i] = [c_i][f_i]$.
   b) each party locally sets $[s_i] = [f_i] + [c_i] - 2[h_i]$ and $[c_{i+1}] = [d_i] + [h_i]$.
4) Set $[s_{\ell+1}] = [c_{\ell+1}]$.
5) Return $[s_1]\ldots[s_{\ell+1}]$.

B. Bit-oriented subtraction

The subtraction operation is used during the division protocol, and we describe a protocol for subtraction of bit-decomposed values explicitly as a modification of the above addition protocol. For notational convenience, we use c_i for $i = 1,\ldots,\ell+1$ to mean the borrow bit (with $c_1 = 0$) and s_i for $i = 1,\ldots,\ell$ to mean the difference bit. We obtain more efficient protocols if the computation is performed on $\neg b_i = 1 - b_i$ rather than b_i, and in what follows we use $\overline{b_i}$ as a shorthand for $\neg b_i$. We obtain:

$$\begin{aligned} s_i &= a_i \oplus b_i \oplus c_i = \neg(a_i \oplus \overline{b_i} \oplus c_i) \\ &= 1 - a_i \oplus \overline{b_i} - c_i + 2(a_i \oplus \overline{b_i})c_i \end{aligned}$$

and

$$\begin{aligned} c_{i+1} &= c_i \wedge \neg(a_i \wedge \overline{b_i}) + \neg c_i \wedge \neg(a_i \vee \overline{b_i}) \\ &= c_i(1 - a_i\overline{b_i}) + (1-c_i)(1 + a_i\overline{b_i} - a_i - \overline{b_i}) \\ &= 1 - a_i - \overline{b_i} + a_i\overline{b_i} + c_i(a_i + \overline{b_i} - 2a_i\overline{b_i}). \end{aligned}$$

This leads to a subtraction protocol below that still uses only $2\ell-1$ multiplications in ℓ rounds.

$\mathsf{Subtract}([a]_B = [a_1]\ldots[a_\ell], [b]_B = [b_1]\ldots[b_\ell])$

1) Each party locally sets $[b'_i] = 1 - [b_i]$ for $i = 1,\ldots,\ell$.
2) For $i = 1,\ldots,\ell$ in parallel execute $[d_i] = [a_i][b'_i]$.
3) Each party locally sets $[f_i] = [a_i] + [b_i] - 2[d_i]$ for $i = 1,\ldots,\ell$ and also $[s_1] = 1 - [f_1]$ and $[c_2] = 1 - [a_1] - [b'_1] + [d_1]$.
4) For $i = 2,\ldots,\ell$,
   a) execute $[h_i] = [c_i][f_i]$.
   b) each party locally sets $[s_i] = 1 - [f_i] - [c_i] + 2[h_i]$ and $[c_{i+1}] = 1 - [a_i] - [b'_i] + [d_i] + [h_i]$.
5) Set $[s_{\ell+1}] = [c_{\ell+1}]$.
6) Return $[s_1]\ldots[s_{\ell+1}]$.

Note that in the division protocol, when subtraction is performed on a and b, we always have that $a \geq b$. This means that we can skip the computation of the last borrow bit $c_{\ell+1}$.

Security. Security follows from the fact that no value is revealed in the clear throughout the execution. This means that, as long as the underlying basic operations provide security against t or less colluding parties, the overall protocol is also secure.