PRIVATE COLLABORATIVE FORECASTING AND BENCHMARKING

by Mikhail Atallah, Marina Bykova, Jiangtao Li, Keith Frikken, and Mercan Topkara

Center for Education and Research in
Information Assurance and Security,
Purdue University, West Lafayette, IN 47907-2086

Private Collaborative Forecasting and Benchmarking[*]

Mikhail Atallah Marina Bykova Jiangtao Li Keith Frikken Mercan Topkara
Computer Sciences Department and CERIAS
Purdue University
{mja,mbykova,jtli,kbf,mkarahan}@cs.purdue.edu

ABSTRACT

Suppose a number of hospitals in a geographic area want to learn how their own heart-surgery unit is doing compared with the others in terms of mortality rates, subsequent complications, or any other quality metric. Similarly, a number of small businesses might want to use their recent point-of-sales data to cooperatively forecast future demand and thus make more informed decisions about inventory, capacity, employment, etc. These are simple examples of cooperative benchmarking and (respectively) forecasting that would benefit all participants as well as the public at large, as they would make it possible for participants to avail themselves of more precise and reliable data collected from many sources, to assess their own local performance in comparison to global trends, and to avoid many of the inefficiencies that currently arise because of having less information available for their decision-making. And yet, in spite of all these advantages, cooperative benchmarking and forecasting typically do not take place, because of the participants' unwillingness to share their information with others. Their reluctance to share is quite rational, and is due to fears of embarrassment, lawsuits, weakening their negotiating position (e.g., in case of over-capacity), revealing corporate performance and strategies, etc. The development and deployment of *private* benchmarking and forecasting technologies would allow such collaborations to take place without revealing any participant's data to the others, reaping the benefits of collaboration while avoiding the drawbacks. Moreover, this kind of technology would empower smaller organizations who could then cooperatively base their decisions on a much broader information base, in a way that is today restricted to only the largest corporations. This paper is a step towards this goal, as it gives protocols for forecasting and benchmarking that reveal to the participants the desired answers yet do not reveal to any participant any other participant's private data. We consider several forecasting methods, including linear regression and time series techniques such as moving average and exponential smoothing. One of the novel parts of this work, that further distinguishes it from previous work in secure multi-party computation, is that it involves floating point arithmetic, in particular it provides protocols to securely and efficiently perform division.

Categories and Subject Descriptors

K.4.4 [**Computers and Society**]: Electronic Commerce—*security*; F.2.m [**Analysis of Algorithm and Problem Complexity**]: Miscellaneous

General Terms

Design, Security

Keywords

Privacy, secure multi-party computation, forecasting, benchmarking, e-commerce, secure protocol

1. INTRODUCTION

One drawback that smaller entities (e.g., individuals, charities, small businesses, etc.) have in competing with large entities (giant corporations and multi-nationals) is that the latter's size and resources enable them to make decisions using more accurate information (e.g., about future demand). This better forecasting ability can, over time, drive the smaller players out and leave the field under the control of the largest entities. Privacy-preserving cooperative computation, which is of obvious benefit to the privacy of individuals, is also a valuable technology for enabling smaller entities to cooperate and make as high-quality decisions as larger entities (decisions about planning, production, quality control, etc.). This paper's focus is on the two specific areas of forecasting of customer demand and secure benchmarking, which are described below. Before we do so, we remind the reader that the broad framework for this work is the usual privacy-preserving computation model, in which two or more parties engage in a collaborative computation in order to produce results that are significant to both parties without revealing the private information of any of the parties, even though the jointly-computed results depend on the information of all the parties.

[*] Portions of this work were supported by Grants IIS-0325345, IIS-0219560, IIS-0312357, and IIS-0242421 from the National Science Foundation, Contract N00014-02-1-0364 from the Office of Naval Research, by sponsors of the Center for Education and Research in Information Assurance and Security, and by Purdue Discovery Park's e-enterprise Center.

WPES'04, October 28, 2004, Washington, DC, USA.

The first problem we are exploring is secure and private collaborative forecasting, in which a number of retailers join their efforts to generate more accurate forecasts of customer demand. We assume that each of the participants has its own proprietary data gathered over some period of time in the past and can produce a local forecast. They decide to participate in joint computation to obtain more reliable results. Consider the following business scenarios:

- A number of small retailers in the area which sell similar products cannot compete with giant stores in their forecasting capabilities. Thus they decide to collaborate with each other in order to better estimate future consumer demand. Revealing data about the past volumes of sales is unacceptable to any of them, as they are competing in the same market. The retailers, however, are willing to share the data in a secure fashion if all that any party learns from the collaboration is the general trend in the customer demand (i.e., increase or decrease in sales and by what amount). After participating in the protocol, each retailer can compare its own locally generated forecast with the large-scale trend, draw conclusions about the accuracy of the local forecast and differences, if any, in the behavior of the sales function at the local and global scopes.

- Another setting where similar collaboration is useful is situations when no single retailer can accurately estimate future demand. Consider a product that has been introduced to the market recently such that no single (even very large) retailer can accurately predict consumer demand for it. This happens when different retailers target differing groups of customers, for which shopping patterns and adaptability to new products vary. Then it is beneficial to all such stores to engage into joint forecasting, while still preserving the privacy of the data on which the forecast is built.

- We can also model a scenario where there is one supplier and many retailers, and the cycle of production is very long. For example, in order for an overseas company to manufacture clothes, it may need to start production 7 months in advance including shipping time. The supplier wants to know the customer demands, i.e., the size of the market. Each retailer is reluctant to provide its own historical data. However, it may benefit the whole supply chain, if the retailers together can collaboratively provide a forecast on customer demands to the supplier. Or, as an alternative, the supplier might provide a discount to all retailers who participate in the joint computation of customer demand, and uses the results for manufacturing more precise quantities.

All of the above scenarios produce forecasts based on time series. Another type of forecasting that we also explore in this work is based on regression techniques. A motivating business model can be as follows. A hospital performs a certain type of surgeries that result in a rather high mortality rate compared to other types of surgeries. The hospital would like to investigate the correlation of the mortality rate to the age of the patients, their health conditions, and possibly other parameters, to be able to exclude the riskiest category of patients from being considered for such surgeries. The hospital, however, does not have enough cases to draw a reliable correlation between the mortality rate and other parameters. The hospital also would like to know how it performs on this kind of surgeries compared to other similar institutions. Thus, the hospital would like to engage in collaboration with other institutions to be able to draw conclusions on the aggregate data, but for privacy reasons cannot share its data with other participants. The solution in this case is to use secure multi-party computation (SMC) techniques that apply regression to aggregate data and distribute the results to all participating parties. Having the results, the hospital then can learn the overall correlation on the large scale, as well as make conclusions about its performance compared to other hospitals.

To address these two problems, we consider forecasting based on time series — moving average, weighted moving average, and exponential smoothing — and regression-based techniques — linear regression. Since the functions used in the computation are linear, some companies might decide that the output of the computation reveals information about their inputs if the number of participants is low (e.g., two). Consequently, they might decide to participate in the computation only if the number of participants is sufficiently large. Therefore, we provide solutions to the problems for a general case of m players.

In this work, we present efficient protocols for conducting secure collaborative forecasting and benchmarking for all statistical methods listed above. Before providing our final protocols, we give sub-protocols, or building blocks, which make presentation of the final protocols crisper and at the same time add flexibility to the protocols themselves by allowing the participants to choose the most appropriate building blocks. In cases when we give more than one protocol for performing the same task, the protocols differ in their complexity, communication overhead, and robustness against colluding players.

A novel part of this work is that we introduce floating point computation in secure multi-party computation. We present several division protocols that form the core of our forecasting and benchmarking solutions, and to the best of our knowledge are the first attempts to perform division without building a generic circuit, as well as the first attempts to operate on floating point numbers. Our division protocols simplify privacy-preserving business forecasting, and can also be applied to other forecasting methods as well as other SMC applications.

A summary of our results is given in table 1. For each protocol described in this work, we list its number of communication rounds, total communication measured in messages exchanged between the players, and total computational complexity (summed over all players). Each message for most protocols is of ℓ bits long, where ℓ is the length of numbers we operate (with the m-key division protocol being an exception). In the table, m refers to the number of players, k is a collusion threshold described in section 4.1 such that $1 \leq k \leq m-1$, and n is a (constant) number of data points used in the linear regression. All of these protocols are later evaluated with respect to the main model of the adversary used in this paper: That of colluding players, i.e., they exhibit the behavior of semi-honest players but can also collude together in order to discover some additional information about other players' data (more on this later). We analyze the collusion-resistance characteristics of each protocol immediately following its description.

Protocol	Communication Rounds	Total Communication	Total Computation
Split	$O(1)$	$O(km)$	$O(km)$
Division with an Appointee	$O(1)$	$O(km)$	$O(km)$
2-party Division with Scaling	$O(\log \ell)$	$O(\log \ell)$	$O(\log \ell)$ encryptions
2-party 2-key Division	$O(1)$	$O(1)$	$O(1)$ encryptions
m-key Division	$O(1)$	$O(m^2)$	$O(m^2)$ encryptions
Moving Average	same as division	same as division	same as division
Exponential Smoothing	same as division	same as division	same as division
Linear Regression	split + n divisions	split + n divisions	split + n divisions

Table 1: Summary of protocols: m is the number of players, k is a collusion threshold where $1 \leq k \leq m-1$, ℓ is the length of numbers in bits, and n is a number of data points in the linear regression.

The rest of this paper is organized as follows. Section 2 reviews related work. In section 3 we briefly provide background information such as different forecasting methods and then provide a more precise definition of our protocols. Section 4 describes building blocks that we developed to aid in designing our main forecasting protocols. The building blocks include a secure algorithm for blinding individual private inputs and — the most interesting and difficult — secure division protocols. Sections 5 and 6 describe our main protocols, where Section 5 covers forecasting based on time series and Section 6 contains regression-based benchmarking. Lastly, Section 7 concludes the paper and provides directions for future work.

2. RELATED WORK

Forecasting is increasingly being applied to business decision making. Many forecasting methods (for example, see [14, 26]) have been developed, such as time-series techniques and regression techniques. Collaborative forecasting allows different entities to jointly perform business forecasting where each entity contributes its own data. As pointed out in [2], collaborative forecasting, in comparison to traditional forecasting, gives better productivity and profitability throughout the supply chain. Collaborative forecasting has been extensively studied by many companies [25, 19], organizations [10], and academia [15]. Most of the solutions either assume existence of a central planner who has all the information about the system, or assume that each participant of the computation shares all of her information with other participants. These solutions, however, are problematic when the data is sensitive and the participants are reluctant to share their private, proprietary information. Our approach is to perform collaborative forecasting in a privacy-preserving manner, therefore eliminates the above concern.

The problem of secure forecasting and benchmarking is closely related to secure multi-party computation [27]. The SMC problem was introduced by Yao [27] and extended by Goldreich, Micali, Wigderson [17] and others ([23, 18], to list a few). Goldreich states in [16] that although the general secure multi-party computation problem is solvable in theory, using the solutions derived by these general results for special cases can be impractical. In other words, efficiency dictates development of special solutions for special cases. And as we can see, many other examples of cooperative privacy-preserving computations have been considered in the literature: electronic auctions [7], card playing [17], digital certified mail, data mining [20], etc.

Du and Atallah recently have developed efficient protocols for many secure *two-party* computation problems [11], including scientific computation [12], geometric computation [5], and statistics analysis [13]. Atallah et al. [6] have proposed Secure Supply-Chain Collaboration (SSCC) problem, and developed SSCC protocols for simple e-Auction scenarios and simple capacity-allocation problem. Our secure collaborative forecasting and benchmarking can be viewed as a branch of the SSCC problem. In this paper, we propose novel protocols for computing a ratio in floating point numbers securely, an important component used in many forecasting techniques. To the best of our knowledge, no one has studied this before.

3. PROBLEM DESCRIPTION

3.1 Background

Before presenting our results, we briefly review several forecasting methods (see [14, 26]) that are the basis of our protocols.

- **Time-Series Techniques.** A time series is a time-ordered sequence of observations taken at regular intervals over a period of time (daily, weekly, monthly, annually, etc.). An example of such data is a monthly estimate of customer demand. Let us here consider a single user environment, where only a local forecast is generated. We use i to denote ith time period, and d_i to denote data in time period i. Let t be the current time period. Using this notation, the three methods that we consider are as follows:

  1. *Moving Average:* Let n denote the number of periods used in calculation of the average. For time period $t \geq n$, the moving average forecast is:

     $$F_t = \Big(\sum_{i=0}^{n-1} d_{t-i}\Big)/n$$

     where F_t indicates the forecasted value for time interval $t+1$.

  2. *Weighted Moving Average:* Let $\vec{w} = \{w_0, w_1, \ldots, w_{n-1}\}$ be a weight vector such that $\sum_{i=0}^{n-1} w_i = 1$. For time period $t \geq n$, the weighted moving average forecast is:

     $$F_t = \sum_{i=0}^{n-1} w_i d_{t-i}$$

3. *Exponential Smoothing:* Let F_i be the forecasted value in time period i, and α be a smoothing constant. For time period t, the exponential smoothing technique computes:

$$F_t = F_{t-1} + \alpha(d_{t-1} - F_{t-1})$$

where F_t is also the predicted value for the next time period.

- **Regression Techniques.** As mentioned above, our regression solutions are built on the most widely used regression method — linear regression: Given two variables with linear correlation, the goal is to compute a linear function such that the sum of the deviations of all the points from the function is minimized. Consider a linear function $y = ax + b$ where all data points x are known. If there are historical data — n pairs of (x, y), then after applying regression to them, we will be able to estimate the coefficients a and b. The coefficients a and b can be computed using the following equations:

$$a = \frac{n(\sum xy) - (\sum x)(\sum y)}{n(\sum x^2) - (\sum x)^2}, \quad b = \frac{\sum y - a \sum x}{n}. \quad (1)$$

3.2 Protocol Definition

Now we define the interfaces of our forecasting protocols. In the definitions below and in the rest of the paper we use the following notation. We assume that there are m players P_1, P_2, ..., P_m engaged in the computation, where $m \geq 2$.

Notation 1. *Any item superscripted with (j) is held by and known only to player P_j. The same item without a superscript mark corresponds to the sum of the items held by all players, which is assumed to be additively split among the players. For example, if we have that player P_j has $x^{(j)}$, then x is equal to $\sum_{j=1}^{m} x^{(j)}$.*

In the first two protocols, which are based on time series, it is undesirable to learn the absolute result: F_t (the forecasted value) might be considered to be revealing too much information because a player can learn his share of the value and possibly some additional information about other players' data. Therefore, instead of providing its absolute value, we output only the slope $\frac{F_t - d_t}{d_t}$, i.e., the percentage by which the value is expected to increase or decrease in the next time interval. Definition 1 corresponds to forecasting based on moving average techniques, and definition 2 is for exponential smoothing forecasts. The detailed protocols are given in section 5.

Definition 1. *Secure Collaborative Forecasting Using Moving Average Techniques*

Input *Player P_j, $1 \leq j \leq m$, provides input data $d_{t-i}^{(j)}$ for n time intervals, where $0 \leq i \leq n-1$. In case of computing the weighted moving average, the weight vector $\vec{w}$ is public.*

Output *Player P_j, $1 \leq j \leq m$, learns $\frac{F_t - d_t}{d_t}$ without anything else, where F_t is computed using the moving average or the weighted moving average technique.*

Definition 2. *Secure Collaborative Forecasting Using Smoothing Techniques*

Input *Player P_j, $1 \leq j \leq m$, supplies $d_{t-1}^{(j)}$, $d_t^{(j)}$, and $F_{t-1}^{(j)}$, where the value of F_{t-1} from the previous time interval computation is kept additively split among all players. The value of α is public.*

Output *Player P_j, $1 \leq j \leq m$, learns $\frac{F_t - d_t}{d_t}$ without anything else, where F_t is computed using the exponential smoothing technique.*

For the linear regression protocols, we assume that the x-axis is public, and the set of possible x values is finite. We use $x_1, x_2, \ldots, x_n$ to denote n possible x-values. In our model, each player supplies the y-axis data and they jointly compute the result in the normalized form. This means that the data, for instance, is given as the average number of accidents per customer in case of car insurance data, or as the mortality rate for surgical cases. In this case each data point y_i is given as two integers c_i and d_i where $y_i = c_i/d_i$. The aggregate values for each data point computed during the execution of the protocol is then found as $y_i = \sum_{j=1}^{m} c_i^{(j)} / \sum_{j=1}^{m} d_i^{(j)}$. The protocol that corresponds to the definition below is provided in section 6.

Definition 3. *Secure Collaborative Benchmarking Using Linear Regression Techniques*

Input *Player P_j, $1 \leq j \leq m$, provides data points $y_i^{(j)}$, where $1 \leq i \leq n$ and each $y_i^{(j)}$ is supplied in the form of $(c_i^{(j)}, d_i^{(j)})$ with $y_i^{(j)} = c_i^{(j)}/d_i^{(j)}$. If P_j does not have data for x_i, then he sets both $c_i^{(j)}$ and $d_i^{(j)}$ to 0.*

Output *Player P_j, $1 \leq j \leq m$, learns the coefficients a and b, such that $\vec{y} = a\vec{x} + b$ where a and b are the cooperatively computed linear regression parameters.*

In terms of the hospital example, suppose m hospitals want to jointly benchmark their mortality rates of surgery operations on heart diseases. x_i then could be the categorized health conditions of the patients, whereas y_i is the overall mortality rates for patients within catalog x_i. We assume there is a linear relation between x (health condition) and y (mortality rate). For each x_i, hospital P_j provides its own mortality rate $y_i^{(j)} = c_i^{(j)}/d_i^{(j)}$, where $c_i^{(j)}$ and $d_i^{(j)}$ are the number of deaths and the number of patients, respectively. y_i is the overall mortality rate for x_i-conditioned patients, therefore it is the sum of deaths divided by the sum of patients in x_i catalog, i.e., $\sum_{j=1}^{m} c_i^{(j)} / \sum_{j=1}^{m} d_i^{(j)}$.

All protocols presented in this work are evaluated in terms of their computational and communication complexity, as well as in the number of communication rounds they require. For the purposes of our evaluation, a communication round is defined as an exchange of messages between the players during which a single player can (i) receive one or more messages from other players, perform calculations, and send one or more messages, *or* (ii) send one of more messages and then receive one or more messages from other players. Every player participates in a single round at most once, with the total number of messages sent over the network during one round being up to m^2.

4. BUILDING BLOCKS

Giving the fully-developed protocols would make them too long and rather hard to comprehend. This section aims at making the later presentation of the protocols crisper by presenting parts of our solutions ahead of time. The building blocks that we describe in this section are an important part of this work, because they lay the ground for solving the forecasting problems in a secure fashion and provide a design choice for the final protocols. This section presents split and division protocols, where the later protocols operate on floating point numbers and thus are new to SMC. We have also developed secure summation and comparison protocols that provide alternative solutions to forecasting based on moving average and weighted moving average techniques. They are not included in this paper due to space constraints but can be found in [4].

We consider three different types of players with respect to malicious behavior:

1. *Semi-honest players:* Semi-honest players (also known as "honest but curious") will follow the protocol as prescribed, but might also attempt to discover more information based on the data they receive at various steps of the protocol.

2. *Colluding players:* Colluding players exhibit the behavior of semi-honest players but can also collude together in order to discover some additional information about other players' data.

3. *Malicious players:* Malicious players may arbitrarily misbehave: they can collude against other players and can deviate from the correct steps of the protocol. Different types of deviation from the protocol include supplying incorrect data, modifying data at intermediate steps of the protocol (possibly in collaboration with other malicious players), prematurely quitting the protocol, or performing incorrect computations at certain steps of the protocol.

In our solutions, we focus on the first two types of players. Considering only semi-honest players is not sufficient because in our case the players might be competing businesses, which does not allow us to exclude colluding behavior from consideration. We do not consider the third type of misbehaving players on the grounds that all the players are interested in the outcome of the computation and will not attempt to disrupt it. Some of our solutions can be tuned to provide a trade-off between complexity and robustness against colluding behavior, and should be setup to account for the expected behavior of the players with respect to collusions. In other words, if during the computation it is not expected that a significant number of players will collude, the protocol can be made more efficient by setting the collusion threshold low.

4.1 Secure split protocol

The first protocol that we present is a secure split protocol that is used as a building block in the final protocols as well as in other building blocks. Prior to execution of the split protocol all players additively share an item where the individual shares are private information. The goal of this protocol is to "blind" individual shares in such a way that no share reveals private information, but the total sum of all shares stays the same as before. At the end of the protocol each player holds a large random number, and the initial private input stays hidden. The details of this protocol are reminiscent of the techniques used in the dining cryptographers [9].

PROTOCOL 1. *Secure Split Protocol*

Input *Player P_j, $1 \leq j \leq m$, provides private input $x^{(j)}$.*

Output *Player P_j, $1 \leq j \leq m$, obtains $z^{(j)}$ such that $\sum_{j=1}^{m} x^{(j)} = \sum_{j=1}^{m} z^{(j)}$.*

Protocol Steps: *1. All players jointly agree on a collusion threshold k, such that $1 \leq k \leq m-1$.*

2. Each player P_j splits $x^{(j)}$ between $k+1$ players in the following way: Player P_j generates k large random values $r_1^{(j)}, \ldots, r_k^{(j)}$ (both positive and negative) and sends them to randomly chosen k players from the remaining $m-1$ players. He sets his share of $x^{(j)}$ to be $r_0^{(j)} = x^{(j)} - r_1^{(j)} - \ldots - r_k^{(j)}$.

3. After receiving k' messages from other players (on average $k' = k$), each player P_j computes $z^{(j)} = r_0^{(j)} + \sum_{i \neq j} r_l^{(i)}$ such that player P_i sent $r_l^{(i)}$ to P_j for some $1 \leq l \leq k$.

Analysis *This method of hiding data is secure but not in an information-theoretic sense. As can be seen from the protocol, a single private input is distributed among $k+1$ players. When this protocol is used as a part of another protocol, individual shares $z^{(j)}$'s are revealed. In order for an individual $x^{(j)}$ to be revealed, however, all of the k players to whom player P_j sent messages in step (2), and all players who sent messages to player P_j in step (3), must collude. Assume that $m' < m$ is the number of players that collude against player P_j. Then the probability of compromising $x^{(j)}$ is 0 when $m' < k$. When $m' \geq k$, the probability of compromise is less than:*

$$\left(\frac{m'}{m-1}\right)^k \left(1 - \frac{k}{m-1}\right)^{m-m'-1}$$

which exponentially decreases as k increases. For instance, when the number of colluders is less than $m/2$ and m is even, the probability of a successful compromise is less than:

$$2^{-k}\left(1 - \frac{k}{m-1}\right)^{\frac{m}{2}}$$

which is upper-bounded by 2^{-k}. If we use $k = c \log m$, where c is a constant, the probability is upper-bounded by m^{-c}. This means that a sub-linear collusion threshold k results in a performance that is probabilistically collusion-resistant against a linear number of colluders.

The protocol is even better with respect to collusive behavior than this probability analysis implies, because the colluders have no way of knowing whether they succeeded or not. Even if the colluders knew from step (2) that all of the k messages that player P_j sent went to the colluding players, they still would not know whether some non-colluding player in step (3) chose player P_j to be a recipient of one of his messages. Thus the colluders have no "success indicator" to tell them whether they succeeded in the compromise or not.

This protocol is performed in 1 round (assume all the players execute the protocol simultaneously); the total communication is $O(km)$ messages; and computational complexity at each player is $O(k)$. When k takes the highest value available $k = m - 1$, a collusion of any number of players less than $m - 1$ has zero probability to succeed. Communication complexity in this case reaches $O(m^2)$.

4.2 Secure division protocols

It is possible to use the general circuit simulation results to carry out secure division, and we need to compare this approach to our approaches. The practical circuits for 2-party ℓ-bit division have size $O(\ell^2)$, and simulation of this circuit requires $O(\ell)$ 1-out-of-2 Oblivious Transfers and $O(\ell^2)$ evaluations of pseudorandom functions (such as AES). Although there are asymptotic improvements to these circuits, they come at the cost of huge constant factors; the asymptotically best of them (and the worst in terms of having impractically large constant factors) is a circuit of size $O(\ell \log \ell \log\log \ell)$ [8, 3] derived from the textbook Schoenhage-Strassen integer multiplication algorithm [24] (which is itself of mainly theoretical interest, and not used in practice). Converting a 2-party division protocol into a general m-party division protocol adds an additional factor of $O(m^2)$.

The protocols we have developed to handle secure multiparty division use a secure multiplication protocol in their various steps, and each such multiplication protocol is easily carried out using $O(1)$ homomorphic encryption computations; we count the number of expensive cryptographic operations (e.g., homomorphic encryption, oblivious transfer) rather than bits communicated because, for both circuit simulations and our protocols, the latter can be obtained from the former by multiplying by ℓ, hence for relative comparisons we can determine which is better using the former.

The protocols we give differ in their constraints on the number of players, communication and computational complexity of the protocols, and their robustness against colluding players. We present them in the order of their simplicity (simplest first). The first protocol operates on floating point numbers. The other three protocols utilize homomorphic encryption and operate on integers, but the result is still computed as a floating point number.

Given two numbers x and y additively split among m players, all of our protocols will output x/y if $y \neq 0$ and a special symbol which indicates that division is not possible when $y = 0$. This means that all players will learn whether $y = 0$, which is unavoidable as it is an inherent part of the answer (hence it is not an information leak).

The idea behind the first protocol is that one of the players is randomly selected to perform division on items that all other players previously blind. This player performs functionality similar to that of an untrusted server and returns the result to all other players, who then unblind it. A version of this protocol that uses an external untrusted server can be found in [4].

PROTOCOL 2. *Secure Division Protocol DIV1*

Input *Player P_j, $1 \leq j \leq m$, provides $x^{(j)}$ and $y^{(j)}$.*

Output *Player P_j, $1 \leq j \leq m$, learns $\frac{x}{y}$.*

Protocol Steps:

1. *All players randomly select one player among all of them who will perform division. Without loss of generality, assume that player P_m is chosen.*
2. *Player P_m splits his items $x^{(m)}$ and $y^{(m)}$ into $m-1$ random numbers each, i.e., $x^{(m)} = \sum_{j=1}^{m-1} r_x^{(j)}$ and $y^{(m)} = \sum_{j=1}^{m-1} r_y^{(j)}$. Player P_m sends each pair $r_x^{(j)}$ and $r_y^{(j)}$ to player P_j.*
3. *Player P_j, $1 \leq j \leq m-1$, receives $r_x^{(j)}$ and $r_y^{(j)}$ from player P_m and sets $x^{(j)} = x^{(j)} + r_x^{(j)}$ and $y^{(j)} = y^{(j)} + r_y^{(j)}$.*
4. *Players P_1 through P_{m-1} engage in the secure split protocol two times providing 0 as their input. Player P_j stores the results of the protocol invocations as $\rho_1^{(j)}$ and $\rho_2^{(j)}$, respectively.*
5. *Players P_1 through P_{m-1} jointly agree on two random floating point numbers α and β.*
6. *Player P_j, $1 \leq j \leq m-1$, computes a pair of values $\langle a^{(j)} = \alpha(x^{(j)} + \rho_1^{(j)}), b^{(j)} = \beta(y^{(j)} + \rho_2^{(j)})\rangle$ and sends the pair to player P_m.*
7. *Player P_m receives $m-1$ pairs $\langle a^{(j)}, b^{(j)}\rangle$, and computes $a = \sum_{j=1}^{m-1} a^{(j)}$, $b = \sum_{j=1}^{m-1} b^{(j)}$, and then $\delta = \frac{a}{b}$. Player P_m sends δ to each of players P_1 through P_{m-1}.*
8. *Player P_j, $1 \leq j \leq m-1$, recovers the value of $\frac{x}{y}$ by computing $\frac{\beta}{\alpha}\delta$. The value of $\frac{x}{y}$ is then sent to player P_m.*

Note: *If it is desired to have the result split among all parties at the end of the protocol, player P_m can split δ into $m-1$ random floating point numbers $\delta^{(j)}$, i.e., $\delta = \sum_{j=1}^{m-1} \delta^{(j)}$ and send each $\delta^{(j)}$ to the corresponding player P_j. Player P_j, $1 \leq j \leq m-1$, then recovers the result by computing $\frac{\beta}{\alpha}\delta^{(j)}$, and splits it into two parts, one of which is kept locally, while the second one (purely random) is sent to player P_m. Player P_m then collects these $m-1$ numbers and sets his share of the result to be their sum.*

Analysis *This protocol works when the total number of players $m \geq 3$. It is resilient to collusions if the player who performs division (P_m) does not collude with other players, but if that player colludes with any other player then the aggregate x and y can be revealed. The probability of learning individual $x^{(j)}$ or $y^{(j)}$ depends on the collusion threshold k used in the split protocol (see the analysis of protocol 1) and is further lowered by the fact that the player who performs division must be among the colluding players.*

This protocol can be performed in 5 rounds, with total communication of $O(km)$, where k is the collusion threshold for the split protocol. The computational complexity for the player who performs division is $O(m)$, and it is $O(k)$ for every other player.

Instead of dedicating a single player for performing division, all players can be divided into two groups where the first group performs division for the second one and the second group performs division for the first group; then result is recovered jointly by both groups. Such protocol can be further generalized to a larger number of groups. See [4] for more details on the protocol.

As the previous protocol only works for $m \geq 3$, next we present solutions to two-party division. In the two-party division protocols and the one following them, all of which use homomorphic encryption, we assume that all players prior to protocol initiation agree on a range of possible values. That is, they define $MAXINT$ to be a large number, such that all possible (aggregate) values of x and y will be less than $MAXINT$, but some randomly generated numbers may exceed $MAXINT$ (in which case this is explicitly stated when they are defined). Also, we consider $1/MAXINT$ to be a negligible error.

Another assumption that we make in these protocols is that both x and y are non-negative numbers. This is an acceptable limitation because all the forecasting methods that we solve operate on positive quantities. Lastly, all encryption arithmetic is integer-based. If players want to provide their inputs as floating point numbers, they need to agree to convert them to integer representation by ignoring decimal points up to a certain precision.

We now give a two-party division a protocol that is provably secure and operates in $O(\log \ell)$ rounds for ℓ-bit numbers, and requires $O(\log \ell)$ homomorphic encryptions and $O(\log \ell)$ oblivious transfers. We summarize what it achieves in the theorem that follows, where by "secure" we mean provably secure in an information-theoretic sense.

THEOREM 1. *If two ℓ-bit numbers x and y are given modularly additively split between two parties P_1 and P_2, then it is possible for the two parties to securely compute the ratio $\theta = x/y$ to within ℓ bits of precision in $O(\log \ell)$ rounds, where each round is done with $O(1)$ homomorphic encryptions and one oblivious transfer.*

Proof Sketch: If we could somehow compute the integer $z = \lfloor 2^{2\ell-1}/y \rfloor$ in split fashion within the claimed bounds, then it would be easy to compute $x * z$ in split fashion by doing one additional split multiplication of x and z; recall that such a split multiplication of two integers can be done in $O(1)$ rounds and using $O(1)$ homomorphic encryption computations. After this, the two parties would exchange their halves of $x * z$ and thereby learn θ to within the desired accuracy. So the main problem is how the two parties can securely compute, in split fashion and to within ℓ bits of accuracy, the integer z. (Note that z has to be computed in split fashion, otherwise both parties can deduce y from it.) Before turning our attention to this computation of z, we observe that we can choose, without loss of security, the modulus of the additive-splitting to be the same as the one for the homomorphic encryption system. That is, if arithmetic is modulo T for the homomorphic encryption, then we can assume that P_1 (resp., P_2) initially has $x^{(1)}$ and $y^{(1)}$ ($x^{(2)}$ and $y^{(2)}$), such that $x^{(1)} + x^{(2)} \bmod T = x$, and $y^{(1)} + y^{(2)} \bmod T = y$. For reasons that will become clear later, we also assume that T is chosen such that $T > 2^{3\ell+1}$.

Because z is the reciprocal of y, the first thought that comes to mind is to use the reciprocal-computation technique based on the centuries-old "Newton's method" (as was done in [1] and [3]). This, however, runs into a subtle difficulty due to the fact that y is split, and the answer z (and all intermediate answers of the iterative process) must also be in split form: Each iteration, when used on integers and producing an integer answer, involves "scaling" (division by a quantity C known to both parties). But because the values to be divided by that C are additively split modulo T, the two parties cannot simply divide their respective shares by this C as it may introduce an error due to the "modulo T" wraparound. The protocol we give below overcomes this difficulty. Before giving the protocol, we recall that the basic idea of the iterative computation of z is to use a sequence of approximations to z that are progressively more accurate: Specifically, if z_i approximates z to within α correct bits, then:

$$z_{i+1} = 2z_i - \lceil (z_i)^2 y / 2^{2\ell-1} \rceil$$

approximates z to within 2α correct bits (the earlier-mentioned scaling factor C is therefore $2^{2\ell-1}$). It will so happen (as will become clear in the split scaling protocol below) that because we operate on split values our protocol's z_{i+1} approximates z_i to within $2\alpha - 2$ bits. Therefore, as long as we start with a split z_0 that approximates z to within a few correct bits, each iteration approximately doubles the number of correct bits in the approximation. This implies that $O(\log \ell)$ iterations suffice (actually, $\log \ell$ if z_0 starts out with 3 correct bits). The above iterative formula actually *always* converges as long as $z_0 \in \{2, 2^\ell - 1\}$: It just converges faster if z_0 starts out with at least a few (e.g., 3) correct most significant bits. There are many ways of computing a good enough initial z_0, and these will be described in the full version of the paper.

Each iteration requires carrying out two split multiplications (hence $O(1)$ homomorphic encryptions) and one split subtraction. Each iteration also requires scaling $w = (z_i)^2 y$ by the factor C to obtain w/C in split fashion, and the protocol for doing so is given below (we dropped the "ceiling" notation in it to avoid unnecessarily cluttering the presentation).

PROTOCOL 3. *Secure Two-Party Protocol for Scaling by a Constant*

Input *Player P_1 has $m^{(1)}$, P_2 has $m^{(2)}$, such that $m^{(1)} + m^{(2)} \bmod T = w$, and $w < 2^{3\ell}$ (because w is the product of three ℓ-bit terms).*

Output *Player P_1 receives updated $m^{(1)}$ and P_2 receives $m^{(2)}$, such that $m^{(1)} + m^{(2)} \bmod T = w/C$.*

Protocol Steps: *1. P_1 chooses a random $r \in [0, T - 1]$ and updates $m^{(1)}$ by doing $m^{(1)} = -r + m^{(1)}/C \bmod T$.*

2. The two players engage in an oblivious transfer protocol in which P_1 prepares a pair (a_0, a_1) and P_2 obtains an $x \in \{a_0, a_1\}$ where

- *$(a_0, a_1) = (r, r - T/C)$ if $m^{(1)} < 2^{3\ell}$*
- *$(a_0, a_1) = (r - T/C, r - T/C)$ if $m^{(1)} \geq 2^{3\ell}$*
- *$x = a_{\{m^{(2)} \geq 2^{3\ell}\}}$, i.e., x is a_1 if $m^{(2)} \geq 2^{3\ell}$ and a_0 otherwise.*

3. P_2 updates $m^{(2)}$ by doing $m^{(2)} = x + m^{(2)}/C$.

Correctness of the split scaling protocol follows from the two possible cases for $m^{(1)} + m^{(2)}$ prior to the update done by the protocol:

- If, before the protocol's updates, $m^{(1)} + m^{(2)} < 2^{3\ell}$ then both $m^{(1)}$ and $m^{(2)}$ are $< 2^{3\ell}$, and the protocol updates by setting $m^{(1)} = -r + m^{(1)}/C$ and $m^{(2)} = r + m^{(2)}/C$, which is correct because the new values $m^{(1)}$ and $m^{(2)}$ result in $m^{(1)} + m^{(2)} \bmod T = w/C$, as required.

- If, before the protocol's updates, $m^{(1)} + m^{(2)} \geq 2^{3k}$ then, because $w = m^{(1)} + m^{(2)} \bmod T = w < 2^{3\ell}$, it must be the case that $m^{(1)} + m^{(2)} \in [T, T + 2^{3\ell}]$. This means that dividing each of $m^{(1)}$ and $m^{(2)}$ by C in this case would introduce an additive T/C error that must be subtracted out by the protocol. To see that it is subtracted out, note that in this case at least one of $\{m^{(1)}, m^{(2)}\}$ is $\geq 2^{3\ell}$, and therefore P_2 is guaranteed to obtain $x = r - T/C$ and the $-T/C$ term in x cancels out the above-mentioned additive error of $+T/C$.

This completes the proof. □

The following is a high-level summary of the protocol described in the above proof.

PROTOCOL 4. *Secure Two-party Division Protocol DIV2*

Input *Player P_1 provides $x^{(1)}$ and $y^{(1)}$, and player P_2 provides $x^{(2)}$ and $y^{(2)}$. Every one of $x^{(1)}$, $y^{(1)}$, $x^{(2)}$, $y^{(2)}$ is in $[0, T)$ where T is the modulus of the homomorphic encryption system used. Both $x = x^{(1)} + x^{(2)} \bmod T$ and $y = y^{(1)} + y^{(2)} \bmod T$ are assumed to be ℓ bits long, hence smaller than 2^{ℓ}. T is chosen such that $T > 2^{3\ell+1}$.*

Output *Players P_1 and P_2 learn $\frac{x}{y}$ to within ℓ significant bits.*

Protocol Steps: *1. Starting with split z_0, for $i = 1, 2, \ldots, 2\log\ell$, P_1 and P_2 securely compute z_i in split fashion according to the iteration equation:*

$$z_{i+1} = 2z_i - \lceil (z_i)^2 y / 2^{2\ell-1} \rceil$$

by using secure split multiplication twice, and using once the split scaling protocol described in the proof of Theorem 1. A known technique for performing secure two-party multiplication is given in Appendix A. As noted earlier, $z = z_{2\log\ell}$ is an ℓ bit integer that equals $\lfloor 2^{2\ell-1}/y \rfloor$.

2. P_1 and P_2 securely multiply z by x in split fashion. The resulting $x * z$ is x/y "scaled up" so it is in integer form. They exchange their shares of $x * z$ and learn x/y to within ℓ significant bits.

Analysis *See the proof of Theorem 1 for computational and round complexity. Communication overhead for each user is $O(1)$ messages per round, and therefore $O(\log\ell)$ messages total.*

Although the above protocol is much better than a circuit simulation approach to solving the secure division problem, it is still expensive. Because the quantities used in forecasting are often private only in an approximate sense, it is worthwhile to consider protocols that "leak" some information but are more efficient (for example, one may not mind if others know the sales figures increased by between 5 and 10 percent, as long as they do not know the exacts percentage).

PROTOCOL 5. *Secure Two-party Division Protocol DIV3*

Input *Player P_1 provides $x^{(1)}$ and $y^{(1)}$, and player P_2 provides $x^{(2)}$ and $y^{(2)}$.*

Output *Both players learn $\frac{x}{y}$, where $x = x^{(1)} + x^{(2)}$ and $y = y^{(1)} + y^{(2)}$.*

Protocol Steps: *1. Player P_1 generates a (public, private) key pair in a homomorphic semantically secure encryption system [21, 22] where arithmetic is modulo N, with $N \geq 2 \cdot MAXINT^2$ (Recall that in such a system $E(a) \cdot E(b) = E(a+b)$, and nothing can be learned about c from $E(c)$.)*

2. Player P_1 computes $E(x^{(1)})$ and $E(y^{(1)})$, and sends them to P_2 along with the public key of the homomorphic encryption system.

3. Player P_2 computes $E(x) = E(x^{(1)}) \cdot E(x^{(2)})$ and $E(y) = E(y^{(1)}) \cdot E(y^{(2)})$.

4. Player P_2 chooses four randoms α_1, α_2, β_1, β_2 from a range $[u, MAXINT - u]$ where u is known to P_2 but not to P_1, and then computes:

$$\begin{aligned} p_1 &= E(x)^{\alpha_1} \mod N = E(\alpha_1 \cdot x) \\ q_1 &= E(y)^{\beta_1} \mod N = E(\beta_1 \cdot y) \\ p_2 &= E(x)^{\alpha_2} \mod N = E(\alpha_2 \cdot x) \\ q_2 &= E(y)^{\beta_2} \mod N = E(\beta_2 \cdot y) \end{aligned}$$

Player P_2 then computes $v = p_1 \cdot q_1 = E(\alpha_1 x + \beta_1 y)$ and $w = p_2 \cdot q_2 = E(\alpha_2 x + \beta_2 y)$ and sends them to player P_1. Note that what is inside the encryption is less than N so there is no "wraparound" due to the modulo N arithmetic.

5. Player P_1 decrypts v and w and gets $D(v) = \alpha_1 x + \beta_1 y$ and $D(w) = \alpha_2 x + \beta_2 y$. He then computes their (floating point) ratio $\delta = (\alpha_1 x + \beta_1 y)/(\alpha_2 x + \beta_2 y)$ and sends it to P_2.

6. Player P_2 computes the ratio $\frac{x}{y}$ as $(\beta_1 - \delta \cdot \beta_2)/(\delta \cdot \alpha_2 - \alpha_1)$ and forwards the answer to P_1.

Analysis *This protocol has a limitation: when $x = 0$, the protocol reveals $\beta_1 y$ and $\beta_2 y$ to player P_1. Player P_1 then can determine possible values of y (using a gcd computation, etc.). Thus, this protocol should not be used when x can take the value of 0, or, alternatively, the coefficients β_1 and β_2 could be constructed in a way to minimize the probability of a successful attack when $x = 0$ (see [4] for more information). As this protocol is designed for two-party computation, there is no need to consider collusion.*

The protocol consists of 2 rounds. Each player needs to perform two encryptions in modular arithmetic. Player P_1 additionally creates a key pair and performs two decrypt operations. Player P_2 also performs a small (constant) number of multiplication and exponentiation operations in modular arithmetic. Thus the overall computational and communication complexity is $O(1)$.

Lastly, we give a protocol that is secure against collusions of up to $m - 1$ players. In what follows, the multiplicative coefficients α and β are implicitly constructed as a product of individual α_j's and β_j's, i.e., $\alpha = \prod_{j=1}^{m} \alpha_j$ and $\beta = \prod_{j=1}^{m} \beta_j$ where α_j and β_j are known only to player P_j.

PROTOCOL 6. *Secure Division Protocol DIV4*

Input *Player P_j, $1 \leq j \leq m$, provides $x^{(j)}$ and $y^{(j)}$.*

Output *Player P_j, $1 \leq j \leq m$, learns $\frac{x}{y}$.*

Protocol Steps: *1. Each player P_j generates a (public, private) key pair E_j and D_j in a homomorphic semantically secure system modulo N_j with $N_j \geq MAXINT^{m+1}$. P_j sends to P_1 the public key E_j and the items $p_j = E_j(x^{(j)})$ and $q_j = E_j(y^{(j)})$. Throughout what follows, even as they get updated, p_j and q_j should be thought of as the encryptions of the "current" (i.e., updated) $x^{(j)}$ and (respectively) $y^{(j)}$.*

2. For $i = 1, \ldots, m$ in turn, the following steps are repeated:

(a) In this step player P_i updates the p_j and q_j other than his own (i.e., with $j \neq i$) that he received (in step 1 if $i = 1$, otherwise from P_{i-1} in the previous iteration of (a)–(c)). He does so as follows. First, P_i creates two random numbers α_i and β_i in the range $[MAXINT/2, MAXINT]$. Next, player P_i generates $m-1$ pairs of random numbers (one pair $a_{i,j}$, $b_{i,j}$ for each other P_j), where each such random is less than $MAXINT^{m+1}$. For each p_j and q_j, $j \neq i$, P_i then computes:

$$\begin{aligned} p_j &= p_j^{\alpha_i} \cdot E_j(a_{i,j}) = E_j(x^{(j)}\alpha_i) \cdot E_j(a_{i,j}) \\ &= E_j(x^{(j)}\alpha_i + a_{i,j}) \\ q_j &= q_j^{\beta_i} \cdot E_j(b_{i,j}) = E_j(y^{(j)}\beta_i) \cdot E_j(b_{i,j}) \\ &= E_j(y^{(j)}\beta_i + b_{i,j}) \end{aligned}$$

which implicitly multiplies $x^{(j)}$ (resp., $y^{(j)}$) by α_i (β_i) and then adds a random to it.

(b) Player P_i now updates his own p_i and q_i by doing

$$\begin{aligned} p_i &= E_i(\alpha_i \cdot D_i(p_i) - \textstyle\sum_{j \neq i} a_{i,j}) \\ q_i &= E_i(\beta_i \cdot D_i(q_i) - \textstyle\sum_{j \neq i} b_{i,j}) \end{aligned}$$

which implicitly multiplies $x^{(i)}$ (resp., $y^{(i)}$) by α_i (β_i) and then subtracts from it a random that "cancels out" the random numbers implicitly added in (a) to the other $x^{(j)}$'s (resp., $y^{(j)}$'s).

Note: *The above decryption and re-encryption of p_i and q_i are not necessary, in the sense that the computation in (b) could have been performed on encrypted items just like the computation of (a) was, but we chose to do the arithmetic on un-encrypted values for efficiency reasons.*

(c) If $i < m$ then player P_i sends all of the p_j and q_j (including his own p_i and q_i), as well as all encryption keys E_j, to P_{i+1}. Otherwise $i = m$ and P_m sends every p_j, q_j pair to the corresponding player P_j who then decrypts them with his private key D_j and obtains his final $x^{(j)}, y^{(j)}$, that is, $x^{(j)} = D_j(p_j)$ and $y^{(j)} = D_j(q_j)$ for all j (including $j = m$).

At the end of the kth iteration of (a)–(e) the sum of the m items $x^{(j)}$ is $(\alpha_1 \cdots \alpha_k x)$ and the sum of the m items $y^{(j)}$ is $(\beta_1 \cdots \beta_k y)$. Therefore at the end of step (1) the sum of the m resulting $x^{(j)}$ is αx. Similarly, the sum of the m resulting $y^{(j)}$ is βy. Note that no player knows (or will know) α or β.

3. Every player P_j generates two random numbers $r_x^{(j)}$ and $r_y^{(j)}$ less than $(MAXINT^{m-1})/(2^{m-1}m)$, and sets $x^{(j)} = x^{(j)} + r_x^{(j)}$ and $y^{(j)} = y^{(j)} + r_y^{(j)}$. Now the sum of all $x^{(j)}$'s will give $\alpha x + r_x$ and the sum of $y^{(j)}$'s is $\beta y + r_y$, where r_x and r_y are negligible compared to αx and βy (more discussion of this follows).

4. All players engage in the secure split protocol providing their $y^{(j)}$ as input and obtaining $y'^{(j)}$. Then every player P_j publishes his share $\rho^{(j)}$.

5. After receiving all the $\rho^{(i)}$'s, each player P_j computes the sum of all the $\rho^{(i)}$'s, which equals $\beta y + r_y$. Since everyone now knows $\beta y + r_y$, each player P_j can compute $\delta^{(j)} = x^{(j)}/(\beta y + r_y)$.

6. Every player P_j reveals to all others the (floating point) ratio $t_j = \beta_j/\alpha_j$.

7. Every player P_j computes $\delta^{(j)} t_1 t_2 \cdots t_m = \delta^{(j)}\beta/\alpha$, which results in the approximation of $\frac{x}{y}$ in additively split form among the m players (with P_j's share being $\delta^{(j)}\beta/\alpha$). To recover the answer as the sum of these shares, they run a secure split protocol, then compute (and reveal to all) the sum of all shares, which is $\frac{x}{y}$ with the necessary precision.

Analysis *The aggregate random numbers r_x and r_y are added to αx and βy to minimize the possibility of factoring αx and βy. For instance, in step (3) all players receive the sum of $y^{(j)}$'s, and without protecting βy with r_y some players might attempt to factor the value. While it is very computationally expensive to factor this number and furthermore, given its factors, not possible to deterministically differentiate between factors of β and y, we still would like to lower the possibility of success as much as possible. Thus, we require that α_j and β_j are at least as large as $MAXINT/2$, which gives us $\alpha x/r_x \geq MAXINT$ and $\beta y/r_y \geq MAXINT$ and is acceptable (recall that we consider $1/MAXINT$ to be a negligible error). Furthermore, we compute:*

$$\frac{\alpha x + r_x}{\beta y + r_y} = \frac{\alpha x}{\beta y}\left(\frac{1 + r_x/\alpha x}{1 + r_y/\beta y}\right) \approx \frac{\alpha x}{\beta y}\left(1 + \frac{r_x}{\alpha x} - \frac{r_y}{\beta y}\right)$$

which converges to $\alpha x/\beta y$ when $r_x \ll \alpha x$, $r_y \ll \beta y$. Now in order to successfully factor βy, an attacker must try all possible r_y, which is a prohibitively large number on the order of $MAXINT^{m-1}$.

This protocol does not scale well to large m's because the length of the numbers that players operate is linear in the number of players. The protocol is conducted in 5 rounds, with the total communication of $O(m^2)$ items (or $O(m)$ messages). The computational complexity at each player is bounded by key generation (which can be precomputed) and $O(m)$ encryptions.

The above protocol provides a high degree of protection where a collusion of any number of players cannot succeed. This might not be required in certain settings, and the protocol can be tuned to lower its robustness and at the same time lower its communication and computational cost. Similar to protocol 1, we can randomly select a subset of k players ($2 \leq k \leq m$) who will conduct the above protocol after all other players distribute their individual shares among those k. By tuning the value of k, the players can find a balance between the acceptable resilience and complexity of the protocol. We do not provide detailed analysis of this protocol here due to space limitations. Other tradeoffs are possible, e.g., decreasing the round complexity at the expense of a higher computation complexity by using circuit simulation. Again, we omit the details.

5. SECURE TIME-SERIES FORECASTING

This section gives final protocols for performing collaborative forecasting based on time series. We start with moving average, then proceed with weighted moving average, and lastly cover exponential smoothing.

5.1 Moving average

The goal of moving average forecasting is to find the behavior of the function at time $t+1$ relative to the current time t. This value can be computed as:

$$\begin{aligned} x &= \frac{F_t - d_t}{d_t} = \frac{\big(\sum_{i=0}^{n-1} d_{t-i}\big)/n - d_t}{d_t} \qquad (2) \\ &= \frac{d_{t-n+1} + \ldots + d_{t-1} - (n-1)d_t}{nd_t} \end{aligned}$$

Below we provide a protocol for solving the moving average problem based on our division protocols. For moving average and weighted moving average, we have developed alternative solutions that use binary search and a secure comparison protocol as their building blocks and can be found in [4]. We do not give their details in this paper.

PROTOCOL 7. *Secure Moving Average Protocol*

Input *Player P_j, $1 \leq j \leq m$, has input data $d_{t-i}^{(j)}$ for n time intervals, where $0 \leq i \leq n-1$.*

Output *Player P_j, $1 \leq j \leq m$, learns $\frac{F_t - d_t}{d_t}$, F_t is computed as the moving average.*

Protocol Steps: *1. Each player P_j sets $x^{(j)} = d_{t-n+1}^{(j)} + \ldots + d_{t-1}^{(j)} - (n-1)d_t^{(j)}$ and $y^{(j)} = nd_t^{(j)}$.*

2. All m players jointly conduct a secure division protocol, with each player P_j providing input $x^{(j)}$ and $y^{(j)}$. The output of the division protocol is the output of this protocol, i.e., $\frac{F_t - d_t}{d_t}$.

Analysis *Both complexity and robustness of this protocol depend on the underlying secure division protocol. Communication and complexity requirements are also those of the division protocol because there is no communication in step (1) and only $O(1)$ computation (n is constant).*

5.2 Weighted moving average

Computation of the weighted moving average is very similar to the previous case of the moving average computation. The difference is that all players agree on a weight vector $\vec{w} = \{w_0, w_1, \ldots, w_{n-1}\}$, which is public. According to the formula for computing the weighted moving average, equation (2) for this case becomes:

$$\begin{aligned} x &= \frac{F_t - d_t}{d_t} = \frac{\big(\sum_{i=0}^{n-1} w_i d_{t-i}\big) - d_t}{d_t} \\ &= \frac{w_0 d_{t-n+1} + \ldots + w_{n-2} d_{t-1} - (1 - w_{n-1})d_t}{d_t} \end{aligned}$$

PROTOCOL 8. *Secure Weighted Moving Average Protocol*

Input *Player P_j, $1 \leq j \leq m$, supplies n data points $d_{t-i}^{(j)}$, where $0 \leq i \leq n-1$.*

Output *Player P_j, $1 \leq j \leq m$, obtains $\frac{F_t - d_t}{d_t}$, where F_t corresponds to joint computation of the weighted moving average.*

Protocol Steps *Very similar to Protocol 7's steps:*

1. Each player P_j sets $x^{(j)} = w_0 d_{t-n+1}^{(j)} + \ldots + w_{n-2} d_{t-1}^{(j)} - (1 - w_{n-1})d_t^{(j)}$ and $y^{(j)} = d_t^{(j)}$.

2. All m players jointly conduct a secure division protocol, where each player P_j supplies input $x^{(j)}$ and $y^{(j)}$. The computation results in the desired value.

Analysis *See analysis of protocol 7.*

5.3 Exponential Smoothing

The formula for exponential smoothing can also be rewritten to simplify joint computation. In the formula below assume α is public, F_{t-1} is calculated during the previous execution of the protocol and is additively split between m players. The goal is then to compute:

$$\begin{aligned} x &= \frac{F_t - d_t}{d_t} = \frac{F_{t-1} + \alpha(d_{t-1} - F_{t-1}) - d_t}{d_t} \\ &= \frac{(1-\alpha)F_{t-1} + \alpha d_{t-1} - d_t}{d_t} \end{aligned}$$

PROTOCOL 9. *Secure Exponential Smoothing Protocol Using Division Protocol*

Input *Player P_j, $1 \leq j \leq m$, provides input data $d_{t-1}^{(j)}$ and $d_t^{(j)}$, as well as the result of the previous execution of the protocol $F_{t-1}^{(j)}$.*

Output *Player P_j, $1 \leq j \leq m$, learns $\frac{F_t - d_t}{d_t}$, where F_t is the result of exponential smoothing computation, and also gets a share $F_t^{(j)}$ of F_t.*

Protocol Steps: *1. Each player P_j sets $x^{(j)} = (1 - \alpha)F_{t-1}^{(j)} + \alpha d_{t-1}^{(j)} - d_t^{(j)}$ and $y^{(j)} = d_t^{(j)}$.*

2. All players jointly execute a secure division protocol, where each player P_j provides $x^{(j)}$ and $y^{(j)}$ as his input. The output of the division protocol is the output of this protocol.

3. Each player P_j sets $F_t^{(j)}$ as $(1-\alpha)F_{t-1}^{(j)} + \alpha d_{t-1}^{(j)}$.

Analysis *The core of this protocol is the underlying division protocol, therefore all complexity and communication analysis, as well as robustness against colluding players is the same as for the division protocol used.*

6. SECURE LINEAR REGRESSION BENCHMARKING

As was mentioned earlier, we apply the linear regression technique to a set of x_i, y_i values, where the number of points n is set in advance. Then each y_i is given in the form of two numbers c_i and d_i, where $y_i = c_i/d_i$ to make it possible to operate on normalized values and guarantee correct outcome. We consider this scenario to be more general than the one where each player provides only his y_i's values. This is because every protocol that solves a problem with y_i values provided in the form of c_i and d_i pairs can be used to solve that problem where y_i is provided as a single value. In our case, if players decide that division is not necessary, they can follow either of the paths below:

(a) They can agree on the values of $d^{(j)}$'s such that $\sum_{j=1}^m d^{(j)} = 1$.

(b) They can omit the step of the protocol where the y_i's are computed using the division protocol and use their original shares of y_i's instead.

Another assumption that we make in this model is that all values of x_i are known to all players and are agreed upon prior to protocol initiation. This means that all of the x_i's values will be used in computation of the regression coefficients even if a player does not have data for all of the points. If, however, none of the players have data for a specific value of x_i, that point must be excluded from the computation. This means that the players learn what data point is being excluded, which is viewed as additional information about other players' input that should be kept secret. Changing the protocol so that it can handle cases where no data is available for a certain point and no player learns this information will result in significantly more complex solutions both in terms of computation and communication. Therefore, we decide to solve this issue in the following way. The protocol starts as usual, and for each data point we compute $y_i = \big(\sum_{j=1}^m c_i^{(j)}\big)/\big(\sum_{j=1}^m d_i^{(j)}\big)$. If it is detected that this division is not possible to perform because all $c_i^{(j)}$, $d_i^{(j)}$ pairs for a specific data point x_i are zero, then the execution is suspended. Each player will be notified that computation cannot be carried out, and they have two options: they can abort the protocol or continue its execution, but in the latter case information about the missing values will be revealed to all players. If all of the players decide to continue, the value of x_i that caused the problem is excluded from the set of possible points and the protocol is restarted. If at least one of the players decides to abort, execution terminates.

To compute the regression coefficients themselves, we use the formulas given in equation (1). Here the value of $\sum_{i=1}^n x_i$ is public and can be computed by each player. Then the equations becomes:

$$a = A\Big(n\sum_{i=1}^n x_iy_i - \big(\sum_{i=1}^n x_i\big)\sum_{i=1}^n y_i\Big),\quad b = \Big(\sum_{i=1}^n y_i/n\Big) - B$$

where A and B are known to all players and can be precomputed, such that $A = 1/\Big(n\big(\sum_{i=1}^n x_i^2\big) - \big(\sum_{i=1}^n x_i\big)^2\Big)$ and $B = \big(a\sum_{i=1}^n x_i\big)/n$ (notice that B can be computed only after a is known as a result of joint computation).

PROTOCOL 10. *Secure Linear Regression Protocol*

Input *Player P_j, $1 \leq j \leq m$, provides a set of pairs $c_i^{(j)}$, $d_i^{(j)}$, where i corresponds to data points x_1, x_2, ..., x_n.*

Output *Player P_j, $1 \leq j \leq m$, learns the coefficients a and b such that $\vec{y} = a\vec{x} + b$.*

Protocol Steps:

1. *All players engage in a secure division protocol n times to compute y_i for $1 \leq i \leq n$, where y_i's remain additively split among all players.*
2. *Each player P_j locally computes $a^{(j)} = A\Big(n\sum_{i=1}^n x_iy_i^{(j)} - \big(\sum_{i=1}^n x_i\big)\sum_{i=1}^n y_i^{(j)}\Big)$.*
3. *All players engage in the secure split protocol with $a^{(j)}$, publish their outputs, and compute the sum $a = \sum_{j=1}^m a^{(j)}$.*
4. *Each player P_j locally computes $b^{(j)} = \sum_{i=1}^n (y_i^{(j)}/n)$.*
5. *All players engage in the secure split protocol with $b^{(j)}$, publish their outputs and each player computes $b = \sum_{j=1}^m b^{(j)} - B$, where B is subtracted by each player separately from computing the sum of $b^{(j)}$'s.*

Analysis *This protocol is as secure against colluding players as its underlying blocks are (i.e., division and split). Communication and computational complexity of the protocol are also bounded by the division and split protocols, where both of them are invoked a constant number of times (the division protocol is executed n times, where the number of points n is constant).*

7. CONCLUSIONS AND FUTURE WORK

In this work, we provided privacy-preserving solutions to collaborative forecasting and benchmarking that can be used to increase the reliability of local forecasts and data correlations, and to conduct the evaluation of local performance compared to global trends. We gave both building blocks and their use in protocols for a number of different forecasting methods based on time-series and regression techniques. The building blocks are general enough to be used in other protocols for forecasting and benchmarking, as well as in other applications. In particular, the division protocols presented in this work, to the best of our knowledge, are the first attempt to perform division in secure multi-party computation as well as to perform computations on floating point numbers.

This work can be extended in a number of ways. Future directions include:

- The model can be extended to other time-series forecasting techniques.
- Along with providing short-range forecasting, we would like to be able to perform long-range forecasts. Long-range forecasts take into account seasonal changes and other long-range patterns.
- We also would like to design protocols to cover other types of regressions for benchmarking collaboration. This will allow us to draw reliable conclusions for different types of data distributions.

- We would like to make some of the protocols provided in this paper more robust against other types of malicious behavior.

Acknowledgments

The authors are grateful to Vinayak Deshpande and Leroy Schwarz for their input on early stages of this work.

8. REFERENCES

[1] A. Aho, J. Hopcroft, and J. Ullman. *The Design and Analysis of Computer Algorithms*. Addison-Wesley, 1974.

[2] K. Allan, M. Stemper, and O. Tucker. Collaborative forecasting. http://e-business.pwcglobal.com/pdf/CollaborativeForecasting.pdf.

[3] H. Alt. Comparing the combinational complexities of arithmetic functions. *Journal of the ACM*, 35(2):447–460, 1988.

[4] M. Atallah, M. Bykova, J. Li, and M. Karahan. Secure collaborative forecasting and benchmarking. Technical Report CERIAS TR 2004–22, Purdue University, 2004.

[5] M. Atallah and W. Du. Secure multi-party computational geometry. In *International Workshop on Algorithms and Data Structures (WADS2001)*, pages 165–179, 2001.

[6] M. Atallah, H. Elmongui, V. Deshpande, and L. Schwarz. Secure supply-chain protocols. In *IEEE International Conference on Electronic Commerce*, pages 293–302, 2003.

[7] O. Baudron and J. Stern. Non-interactive private auctions. In *Financial Crypto'01*. Springer–Verlag, 2001.

[8] P. Beame, S. Cook, and H. Hoover. Log depth circuits for division and related problems. In *Annual IEEE Symposium on Foundations of Computer Science*, pages l–6, 1984.

[9] D. Chaum. The dining cryptographers problem: Unconditional sender and recipient untraceability. *Journal of Cryptology*, 1(1):65–75, 1988.

[10] Collaborative planning, forecasting, and replenishment (CPFR). http://www.cpfr.org/Members.html.

[11] W. Du. *A Study of Several Specific Secure Two-party Computation Problems*. PhD thesis, Purdue University, 2001.

[12] W. Du and M. Atallah. Privacy-preserving cooperative scientific computations. In *IEEE Computer Security Foundations Workshop*, pages 273–282, 2001.

[13] W. Du and M. Atallah. Privacy-preserving statistical analysis. In *Annual Computer Security Applications Conference*, pages 102–110, 2001.

[14] J. Evans. *Applied production and operations management*. West Publishing Company, 4th edition, 1993.

[15] G. Fliedner. Collaborative planning, forecasting, and replenishment in the retail supply chain. Decision and Information Sciences Department, Oakland University.

[16] O. Goldreich. Secure multi-party computation. http://www.wisdom.weizmann.ac.il/home/oded/public_html/pp.html, 2001.

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

[18] S. Goldwasser. Multi-party computations: Past and present. In *Annual ACM Symposium on Principles of Distributed Computing*, August 1997.

[19] John galt solution, inc. http://www.johngalt.com/.

[20] Y. Lindell and B. Pinkas. Privacy preserving data mining. In *Advances in Cryptology – CRYPTO'00*, pages 36–54, 2000.

[21] D. Naccache and J. Stern. A new cryptosystem based on higher residues. In *ACM Conference on Computer and Communications Security*, pages 59–66, 1998.

[22] T. Okamoto and S. Uchiyama. A new public-key cryptosystem as secure as factoring. In *Advances in Cryptology – EUROCRYPT'98*, volume 1403 of *LNCS*, pages 308–318, 1998.

[23] T. Rabin and M. Ben-Or. Verifiable secret sharing and multiparty protocols with honest majority. In *Annual ACM Symposium on Theory of Computing*, pages 73–85, 1989.

[24] A. Schoehage and V. Strassen. Schnelle multiplikation grosser zahlen. *Computing*, 7:281–292, 1971.

[25] H. Singh. Collaborative forecasting. http://www.supplychain.com/docs/collaborativeforecasting.pdf, 2002.

[26] W. Stevenson. *Production/Operations Management*. Richard D. Irwin, Inc., 4th edition, 1993.

[27] A. Yao. Protocols for secure computations. In *Annual IEEE Symposium on Foundations of Computer Science*, 1982.

Appendix A

It is well known ("folklore") that homomorphic encryption can be used by two parties P_1 and P_2 to carry out secure split multiplication. This is because, if T is the modulus for the homomorphic encryption system, and if all arithmetic is henceforth assumed to be modulo T, then we have $E(a)^b = E(a * b)$. This implies a split multiplication protocol in the following way. Assume x and y are modularly additively split between P_1 and P_2 as $x = x^{(1)}+x^{(2)}$ and $y = y^{(1)}+y^{(2)}$, and let z be the desired answer to be obtained additively split as $z = z^{(1)} + z^{(2)}$. Then

$$xy = x^{(1)}y^{(2)} + x^{(2)}y^{(1)} + x^{(1)}y^{(1)} + x^{(2)}y^{(2)}$$

The last two terms in the above can be computed locally by P_1 (third term) and P_2 (fourth term). The first two terms are computed by having P_1 send to P_2 both $E(x^{(1)})$ and $E(y^{(1)})$, then P_2 (who can encrypt but does not have the private decryption key) chooses a random r and computes:

$$v = E(x^{(1)})^{y^{(2)}})E(y^{(1)})^{x^{(2)}})E(-r) = E(x^{(1)}y^{(2)}+x^{(2)}y^{(1)}-r).$$

and sends v to P_1 who decrypts it and sets $z^{(1)}$ equal to

$$z^{(1)} = D(v) + x^{(1)}y^{(1)} = x^{(1)}y^{(2)} + x^{(2)}y^{(1)} - r + x^{(1)}y^{(1)}.$$

whereas P_2 sets $z^{(2)}$ equal to

$$z^{(2)} = r + x^{(2)}y^{(2)}$$

Note that $z^{(1)} + z^{(2)} = xy$, as required.