CERIAS Tech Report 2004-22

PRIVATE COLLABORATIVE FORECASTING AND BENCHMARKING

by Mikhail Atallah, Marina Bykova, Jiangtao Li, Mercan Karahan

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 Mercan Karahan
Computer Sciences Department and CERIAS
Purdue University

{mja,mbykova,jtli,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.

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

*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.

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 giving smaller entities a fighting chance by enabling them to cooperate and make as high-quality decisions as larger entities (decisions about planning, production, management decisions, 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 (each in turn). 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.

The first problem we are exploring is secure collaborative forecasting, in which a number of retailers join their efforts to generate more accurate forecasts of customer demand in a secure fashion. 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.

In [11], forecasting is likened to driving a car blindfolded while taking directions from someone in the back who is looking in the rearview mirror. What we want to do by secure collaborative forecasting is joining view angles of more rearview mirrors. In this metaphor rearview mirrors are used to say that most of the quantitative forecasting methods use historical data to produce forecasts. Such historical data is private for every company and should not be shared with its competitors. 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 the sales and by what amount). After participating in the protocol, each retailer can compare its own locally generated forecast with a large scale trend, draw its own conclusions about the accuracy of the local forecast and the differences, if any, in the behavior of the sales function at the local and global scopes.

- Another situation when the same kind of collaboration is useful is the cases where no single retailer can accurately estimate the future demand. If some (even very large) retailers attempt to predict consumer demand for a new product, it is possible that no single forecast will be very accurate. This happens when the retailers target different groups of customers, for which shopping patterns and adaptability to new products differ. Then it is beneficial for all of such stores to engage in joint forecasting, while still preserving the privacy of the data on which the forecast is built.

- We can as well 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 clothes company to manufacture clothing, it may need to start seven 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 of the computations for manufacturing more precise quantities.

All of the above scenarios are for producing 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 kind of surgeries that result in a rather high mortality rate compared to other kinds of surgeries. The hospital would like to investigate 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 this type of surgery. The hospital, however, does not have enough such cases to draw reliable correlation between the mortality rate and other parameters. The hospital would like to engage in collaboration with other similar institutions to be able to draw conclusions on the aggregate data, but for obvious reasons cannot share its data with other parties. The solution in this case is to use secure multi-party computation 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.

As mentioned above, we consider two types of forecasting methods: based on time series (moving average, weighted moving average, and exponential smoothing) and regression-based (linear regression). Since the functions we are computing are linear, some companies might decide to participate only if the number of parties involved in the protocol is large. This is because if there are only a few players (e.g., two), then the players might consider the output to be revealing too much information about the players' private data. Thus, we solve 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 of the above-mentioned statistical methods. Before providing the final protocols, we give sub-protocols, or building blocks, which make the final protocols more flexible and their presentation crisper. In some cases, we give more than one protocol for performing the same task where such 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 into the realm of SMC. We present several division protocols that form the core of our forecasting and benchmarking solutions. These protocols significantly simplify privacy-preserving business forecasting, and can also be applied to other forecasting methods and other secure business and non-business 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). In the table, m refers to the number of players, k is the security parameter described in section 4.1 such that $1 \leq k \leq m-1$, and g is the number of groups in which the m players are divided. 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.

The rest of this paper is organized as the following: Section 2 reviews related work. In section 3 we briefly provide the 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 secure

Protocol	Communication Rounds	Total Communication	Total Computation
Split	$O(1)$	$O(km)$	$O(km)$
Division with External Server	$O(1)$	$O(km)$	$O(km)$
Division with Groups	$O(g)$	$O(kgm)$	$O(kgm)$
Division with an Appointee	$O(1)$	$O(km)$	$O(km)$
Single-key Homomorphic Encryption Division	$O(1)$	$O(m)$	$O(m)$ encryptions
m-key Homomorphic Encryption Division	$O(1)$	$O(m^2)$	$O(m^2)$ encryptions
Weak Summation	$O(1)$	$O(m)$	$O(m)$
Robust Summation	$O(1)$	$O(m^2)$	$O(m^2)$
Comparison	$O(1)$	$O(m)$	$O(m)$
Moving Average with Division	same as division	same as division	same as division
Moving Average with Binary Search	same as comparison	same as comparison	same as comparison
Exponential Smoothing	same as division	same as division	same as division
Linear Regression	summation + division	summation + division	summation + division

Table 1: Summary of protocols.

algorithms for blinding individual private inputs, summation of a number of elements, comparison of elements, and — the most difficult — secure division protocol. Sections 5 and 6 describe our main protocols, where section 5 covers forecasting based on time series and section 6 contains regression-based forecasting. Lastly, section 7 concludes the paper and provides directions for future work.

2 Related Work

Forecasting is increasingly being applied to the procedure of business decision making. Many forecasting methods (for example, see [9, 21]) have been developed, such as time-series techniques and regression techniques. Collaborative forecasting is to conduct business forecasting jointly by many entities, each contributes its own forecast data. As pointed out in [1], collaborative forecasting, compares to traditional forecasting, gives better productivity and profitability throughout the supply chain. Collaborative forecasting has been extensively studied by many companies [20, 16], organizations [5], and academia [10]. Most of the solutions either assume existence of a central planner who has all the information about the system, or assume each participant of the collaborative forecasting shares all his information to the others. However, these solutions are problematic when the information is sensitive, and the participants are reluctant to share their private, proprietary information. Our solution to collaborative forecasting is conducted in a privacy-preserving manner, therefore overcoming the above concerns.

The problem of secure forecasting and benchmarking is closely related to secure multi-party computation [23]. The SMC problem was introduced by Yao [23] and extended by Goldreich, Micali, Wigderson [14] and others ([18, 15], to list a few). Goldreich states in [13] that although the general SMC problem is solvable in theory, using the solutions derived by these general results for special cases can be impractical. In other words, efficiency dictates the development of special solutions for special cases for efficiency reasons. In particular, many other examples of cooperative

privacy-preserving computations have been considered in the literature: electronic auction [4], card playing [14], digital certified mail, data mining [17], etc.

Du and Atallah recently have developed efficient protocols for many secure two-party computation problems [6], including scientific computation [7], geometric computation [2], and statistics analysis [8]. Atallah et al. have proposed Secure Supply-Chain Collaboration (SSCC) problem, and developed SSCC protocols for simple e-Auction scenarios and simple capacity-allocation problem [3]. 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 in many forecasting techniques. To the best of our knowledge, no one has studied it before.

3 Problem Description

3.1 Background

Before presenting our results, we briefly review several forecasting methods (see [9, 21]) 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 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 similarly the predicted value for the next time period.

- **Regression Techniques.** As was mentioned above, our regression solutions are built on the most widely used regression method — linear regression. It can be formulated as the following: 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 about x–y pairs, 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:

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

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 $m \geq 2$. 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 might be undesirable to learn the absolute result: F_t may 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., how much 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 protocols themselves 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 the case of computing the weighted moving average, the weight vector $\vec{w}$ is public.

Output Player P_j, $1 \leq j \leq m$, learns $\frac{F_t - d_t}{d_t}$, 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 $\frac{F_t - d_t}{d_t}$, 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 integer numbers 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 itself for 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 $(c_i^{(j)}, d_i^{(j)})$. 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 $\vec{y} = a\vec{x} + b$.

4 Building Blocks

Giving the full-developed protocols would make them too long and rather hard to comprehend. This section aims at making the later presentation of the protocols much crisper by presenting parts of our solutions and building blocks for them 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 make our final protocols much more straightforward. In particular, this section presents division protocols which operate on floating point numbers and thus are new to SMC. Other protocols given in this section are blinding, summation, and comparison.

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:* In this case, 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 protocols we focus on the first two types of players. Considering only the first type of players is not sufficient because players in our case can be competing businesses, which does not allow us to exclude colluding behavior. We, however, do not consider the third type of misbehaving players on the assumption that all 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. In other words, if during the computation it is not expected that a large number of players will collude, the protocol can be made more efficient by setting the security parameter 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 many protocols throughout this paper, including other building blocks. Prior to the execution of the split protocol all players additively share an item but 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 stay the same as before. At the end of the protocol each player holds a large random number, which hides the initial private input.

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 $y^{(j)}$ such that $\sum_{j=1}^{m} x^{(j)} = \sum_{j=1}^{m} y^{(j)}$.

Interaction The steps of the protocol are as given below:

1. All players jointly agree on a security parameter k, such that $1 \leq k \leq m-1$.
2. Each player P_j splits $x^{(j)}$ between all $k+1$ 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. Player P_j then 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 $y^{(j)} = r_0^{(j)} + \sum_{i \neq j} r_l^{(i)}$ 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 in practice, but not in the 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 $y^{(j)}$'s are revealed. In order for an individual $x^{(j)}$ to be revealed, however, all of the k players to whom player P_j sends messages in step (2) and all players who send 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 $m' < k$. Then $m' \geq k$, the probability of compromise is less than:

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

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

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

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 security parameter k results in a performance that is probabilistically collusion-resistant against a linear number of participants.

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 know from step (2) that all of the k messages that player P_j sent went to the colluding players, they still do not know whether some non-colluding player in step (3) chose player P_j to be a recipient one of his messages. Thus the colluders have no knowledge of telling whether they succeeded in the compromise or not.

This protocol is performed in 1 round; the total communication is $O(km)$ messages; and computational complexity at each player is $O(k)$. When k takes the highest value available $k =$

$m-1$, any number of colluders less then $m-1$ have zero probability of success. Communication complexity in this case is, however, $O(m^2)$.

4.2 Secure division protocols

We have developed different protocols to handle secure multi-party division, which operate on floating point numbers. The difference between the protocols is in the number of players used, their computational complexity, and robustness against colluding players. The first protocol provided here uses an external untrusted party to aid in computation, while other protocols do not require any additional servers.

4.2.1 Division protocol using an untrusted external server

This protocol achieves efficiency at the cost of requiring an untrusted third party, to whom we refer as Ursula.

Protocol 2 *Secure Division Protocol with Ursula*

Input Player P_j, $1 \leq j \leq m$, supplies two data items $x^{(j)}$ and $y^{(j)}$.

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

Interaction This protocol has the following steps:

1. Each player P_j splits his value of $x^{(j)}$ into two parts in the following manner: player P_j generates a random number $r^{(j)}$. He sets $x_1^{(j)} = r^{(j)}$ and $x_2^{(j)} = x^{(j)} - r^{(j)}$.
2. All players engage in the secure split protocol three times providing 0 as their private input. Each player P_j stores his output from the first two protocol invocations as $r_{x_1}^{(j)}$ and $r_{x_2}^{(j)}$, respectively, and the output from the last round as $r_y^{(j)}$.
3. All players jointly choose three random floating point numbers α_1, α_2, and β.
4. Each player P_j sends $\langle a_1^{(j)} = \alpha_1(x_1^{(j)} + r_{x_1}^{(j)}), a_2^{(j)} = \alpha_2(x_2^{(j)} + r_{x_2}^{(j)}), b^{(j)} = \beta(y^{(j)} + r_y^{(j)})\rangle$ to Ursula. Observe that $\sum_{j=1}^{m} a_1^{(j)} = \alpha_1 \sum_{j=1}^{m} x_1^{(j)}$, $\sum_{i=1}^{m} a_2^{(j)} = \alpha_2 \sum_{i=1}^{m} x_2^{(j)}$, and $\sum_{j=1}^{m} b^{(j)} = \beta y$.
5. Ursula receives $\langle a_1^{(j)}, a_2^{(j)}, b^{(j)}\rangle$ from each player and computes $a_1 = \sum_{j=1}^{m} a_1^{(j)}$, $a_2 = \sum_{j=1}^{m} a_2^{(j)}$, and $b = \sum_{i=1}^{m} b^{(j)}$.
6. Ursula computes $\delta_1 = \frac{a_1}{b}$ and $\delta_2 = \frac{a_2}{b}$, and sends δ_1 and δ_2 to all players. If Ursula detects that $b = 0$, it sends all players a special symbol which indicates that the value is undefined.
7. Each player computes $\beta(\frac{\delta_1}{\alpha_1} + \frac{\delta_2}{\alpha_2})$ and recovers the value of $\frac{x}{y}$.

Analysis Everything that Ursula learns during the execution of this protocol is whether $y = 0$. This, however, cannot be prevented from happening because in order to perform the computation correctly, Ursula needs to know whether division is possible or not. Notice that Ursula does not learn anything about the value of x because it is split into two parts, which are protected by two different random values α_1 and α_2. Even if one of a_1 and a_2 happens to have the value of 0, this information does not reveal anything about the actual value of x.

This protocol is also secure against collusions as long as no player P_1 through P_m colludes with Ursula, even if $m-1$ players collude against the other player. If, on the other hand, a collusion between the players and Ursula is possible, then some information can be leaked. In particular, if a player colludes with Ursula, the values of x and y can be revealed. But in order to reveal the value of a single $x^{(j)}$ or $y^{(j)}$, a collusion of Ursula and a sufficient number of players is necessary to make the probability of success in protocol 1 feasible (see analysis in protocol 1 for more information).

The protocol is carried out in two rounds. Total communication requirements are $O(km)$ messages, where k is the security parameter used in the split protocol; and computational complexity at each player is $O(k)$.

The above division protocol announces $\frac{x}{y}$ to all players. However, in some situations, we want to keep the result $\frac{x}{y}$ split between m players. Note that the above protocol can be easily converted into a division protocol with the result being split as follows: In Step (6) of the protocol, instead of sending δ_1 and δ_2 to every player, Ursula chooses $\delta_1^{(1)}, \ldots, \delta_1^{(m)}$ such that $\delta_1 = \sum_{j=1}^{m} \delta_1^{(j)}$, and $\delta_2^{(1)}, \ldots, \delta_2^{(m)}$ such that $\delta_2 = \sum_{j=1}^{m} \delta_2^{(j)}$. Ursula then sends $\delta_1^{(j)}$ and $\delta_2^{(j)}$ to each player P_j. Each player P_j computes $s_y(\frac{\delta_1^{(j)}}{\alpha_1} + \frac{\delta_2^{(j)}}{\alpha_2})$, which results in $\frac{x}{y}$ being distributed among m players.

4.2.2 Division protocols without external servers

All division protocols given in this section do not use any external (untrusted) servers but perform all computation inside the group of m players. They, however, differ in their complexity and robustness against colluding behavior.

The next protocol we provide is similar to the one used to perform division with Ursula in the previous section. The difference is that one of the participating players is selected to perform the same functionality as Ursula in the fist division protocol.

Protocol 3 *Secure Division Protocol without External Parties 1*

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

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

Interaction The protocol proceeds in the following 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 values of $x^{(m)}$ 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 the values of $r_x^{(j)}$ and $r_y^{(j)}$ to player P_j, where $1 \leq j \leq m-1$.
3. Player P_j, $1 \leq j \leq m-1$, receives two numbers $r_x^{(j)}$ and $r_y^{(j)}$ from player P_m and adds them to his value of $x^{(j)}$ and $y^{(j)}$, respectively. This results in new values $x^{(j)} = x^{(j)} + r_x^{(j)}$ and $y^{(j)} = y^{(j)} + r_y^{(j)}$ being held by player P_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 $r_1^{(j)}$ and $r_2^{(j)}$, 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 $\langle a^{(j)} = \alpha(x^{(j)} + r_1^{(j)}),\ b^{(j)} = \beta(y^{(j)} + r_2^{(j)})\rangle$ and sends it to player P_m.
7. Player P_m receives $m-1$ pairs of values $\langle a^{(j)}, b^{(j)}\rangle$, 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 players P_1 through P_{m-1}.
8. Player P_j, $1 \leq j \leq m-1$, computes the value of $\frac{x}{y}$ by computing $\frac{\beta}{\alpha}\delta$. The value of $\frac{x}{y}$ is also sent to player P_m.

Analysis In order to have the result split among all parties at the end of the protocol, player P_m splits the result δ into $m-1$ random floating point numbers $\delta^{(j)}$ and sends each of them 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 random parts, one of which is kept locally, while the second one is sent to player P_m.

This protocol works when the total number of players $m \geq 3$. The protocol is robust against collusions if the player who performs division does not collude with other players. However, if that player colludes with another player, the aggregate x and y can be revealed. The probability of learning individual $x^{(j)}$ or $y^{(j)}$ depends on the security parameter k used in the split protocol (see analysis of protocol 1 for more information) and is further lowered by the fact that the player who performs the division must be among the colluding players.

Notice that the player who performs division learns whether $x = 0$ earlier than all others. If that player is not trusted to return the result of the computation to all other players (i.e., he may quit prematurely), this protocol can be modified to split all $x^{(j)}$'s into two parts, similar to Protocol 2. There is no information leakage during the execution of this protocol when $x = 0$, because all players learn this information at the end of the protocol.

This protocol can be performed in 4 rounds, with total communication of $O(km)$, where k is the security parameter for the split protocol. Computational complexity at the player who performs division is $O(m)$, and it is $O(k)$ at any other player.

The next protocol that we provide also delegates internal players to perform division, but now all players are divided into two groups for that purpose. All players jointly compute $\left(\alpha_1\alpha_2 \sum_{j=1}^{m} x^{(j)}\right) / \left(\beta_1\beta_2 \sum_{j=1}^{m} y^{(j)}\right)$, where α_1 and β_1 are known to the first group, and α_2 and β_2 are known to the members of the second group. The data is kept split during the intermediate steps so that no individual can learn extra information from the execution of the protocol.

Protocol 4 *Secure Division Protocol without External Parties 2*

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

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

Interaction The protocol's steps are as the following:

1. The m players are divided into two groups 1 and 2 of $m' = \frac{m}{2}$ members each[1]. Without loss of generality, assume that players P_1 through $P_{\frac{m}{2}}$ become the members of the first group, while players $P_{\frac{m}{2}+1}$ through P_m become the members of the second group.
2. Group 1 agree on a floating point random number α_1 and engage in the secure split protocol. Each group member P_j provides $\alpha_1 x^{(j)}$ as his input to the protocol and stores the output as $a_1^{(j)}$. This results in $\alpha_1(x^{(1)} + x^{(2)} + \ldots + x^{(m')})$ being split among all members of the group.

   Group 2 performs the same steps, which results in $\alpha_2(x^{(m'+1)} + x^{(m'+2)} + \ldots + x^{(m)})$ being distributed among m' players in the form of $a_2^{(j)}$.
3. Groups 1 and 2 repeat the same process for y. This results in $\beta_1(y^{(1)} + y^{(2)} + \ldots + y^{(m')})$ being distributed among the members of the first group in the form of $b_1^{(j)}$; and $\beta_2(y^{(m'+1)} + y^{(m'+2)} + \ldots + y^{(m)})$ among the members of the second group in the form of $b_2^{(j)}$.
4. All members of group 1 engage in the secure split protocol providing 0 as their input. Group 1 member P_j stores his output as $r_{1x}^{(j)}$. Notice that this gives us $\sum_{j=1}^{m'} r_{1x}^{(j)} = 0$, such that no individual $r_{1x}^{(j)}$'s are known to other players. Group 2 performs the same computation, which results in $r_{2x}^{(j)}$ being stored at group 2 member P_j, where $\sum_{j=m'+1}^{m} r_{2x}^{(j)} = 0$.
5. All members of groups 1 and 2 jointly agree on a mapping between a single member of group 1 and a single member of group 2.
6. Each member P_j of the first group sends his share $a_1^{(j)}$ to the corresponding member of the second group; assume this is player $P_{m'+j}$. Member $P_{m'+j}$ computes $c_1^{(j)} = \alpha_2 a_1^{(j)} + r_{2x}^{(m'+j)}$ and sends it back to P_j.

   This action is also performed in the other direction. More precisely, each member of the second group $P_{m'+j}$ sends his share $a_2^{(m'+j)}$ to the corresponding member of the first group (assume P_j) and receives back $c_2^{(m'+j)} = \alpha_1 a_2^{(m'+j)} + r_{1x}^{(j)}$.
7. The previous step is repeated with the values of b. A set of new random numbers $r^{(j)}$'s is generated within each group, i.e., randoms $\sum_{j=1}^{m'} r_{1y}^{(j)} = 0$ and $\sum_{j=m'+1}^{m} r_{2y}^{(j)} = 0$ are selected by and known to the members of the first and second group, respectively. Each member P_j of the first group sends $b_1^{(j)}$ to member $P_{m'+j}$ of the second group and receives back $d_1^{(j)} = \beta_2 b_1^{(j)} + r_{2y}^{(m'+j)}$. Each member $P_{m'+j}$ of the second group sends $b_2^{(m'+j)}$ to member P_j of the first group and receives back $d_2^{(m'+j)} = \beta_1 b_2^{(m'+j)} + r_{1y}^{(j)}$.
8. Two members P_j and $P_{m'+j}$ from the first and the second groups, respectively, compute $d_1^{(j)} + d_2^{(m'+j)}$ and publish the sum. Then each member from either group collects all published values and computes $D = \sum_{j=1}^{m'} d_1^{(j)} + \sum_{j=m'+1}^{m} d_2^{(j)}$.

[1]This can also be handled for odd m's. For instance, one randomly chosen player can split his input among all other $m-1$ players, and they proceed with the protocol as the number of players was even.

9. Each member P_j of groups 1 and 2 computes $e_1^{(j)} = c_1^{(j)}/D$ and $e_2^{(j)} = c_2^{(j)}/D$, respectively. Now they have the results of $\left(\alpha_1\alpha_2 \sum_{j=1}^{m} x^{(j)}\right)/\left(\beta_1\beta_2 \sum_{j=1}^{m} y^{(j)}\right)$ additively split between m players. To recover the output, they sum all $e_1^{(j)}$'s and $e_2^{(j)}$'s together, announce the values of $\frac{\beta_1}{\alpha_1}$ and $\frac{\beta_2}{\alpha_2}$, and multiply the sum by $\frac{\beta_1\beta_2}{\alpha_1\alpha_2}$.

Analysis This protocol requires at least two players per group, i.e., the number of players $m \geq 4$, in order to be secure against non-colluding players. But if a player from group 1 colludes with a player from group 2, the protocol can leak partial information. More precisely, if all values of α's and β's are exchanged by colluding players, the total values of x and y can be computed. However, in order to learn an individual value $x^{(j)}$ or $y^{(j)}$, it is necessary to have a coalition of a significant number of members (defined based on the security parameter k, see analysis in protocol 1 for more information) of the group to which player P_j belongs and the corresponding player from the other group (i.e., in the description above, $P_{m'+j}$ if $j \leq m'$, and $P_{j-m'}$ if $j > m'$).

We can make this protocol more robust with respect to colluding behavior by using more groups. Then in order to obtain aggregate x or y, a malicious player will need to collude with a member of each group. In this case, however, the complexity of the protocol will be increased. That is, if we have g groups, then the players will need to compute:

$$\frac{\alpha_1\alpha_2\cdots\alpha_g \sum_{j=1}^{m} x^{(j)}}{\beta_1\beta_2\cdots\beta_g \sum_{j=1}^{m} y^{(j)}}$$

The protocol proceeds in $O(g)$ rounds, and its total communication is $O(kgm)$ messages, where k is the security parameter for the secure split protocol and g is the number of groups. Computational complexity at each player is $O(kg)$.

Next, we present another division protocol that uses homomorphic encryption and works when the number of players m is as low as two. In this and the next protocol, both of which use homomorphic encryption, 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 and randomly generated numbers will not exceed *MAXINT*. 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, which is a minor limitation because all forecasting methods that we solve operate positive quantities. Lastly, all encryption arithmetic is integer-based. If players want to provide their inputs as floating point numbers, they need to convert them to integer representation by ignoring decimal points up to a certain precision.

Protocol 5 *Secure Division Protocol without External Parties 3*

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

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

Interaction The protocol is conducted through the following steps:

1. All players randomly chose two players among them to carry computation on behalf of the group. Without loss of generality, assume that players P_1 and P_m are chosen.
2. Player P_1 generates a (private, public) key pair in a homomorphic semantically secure encryption system 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 $E(c)$.)
3. Player P_1 creates a list of all players and crosses out his name from it. P_1 sends the public key (i.e., $E(.)$, which includes N) to a randomly chosen player from the list (say P_2 is chosen), together with $E(x^{(1)})$, $E(y^{(1)})$ and the list. Player P_2 crosses out his name from the list, computes $E(x^{(1)}) \cdot E(x^{(2)}) = E(x^{(1)} + x^{(2)})$ and $E(y^{(1)}) \cdot E(y^{(2)}) = E(y^{(1)} + y^{(2)})$, and forwards the list, the results of computation, as well as the encryption key to another randomly chosen player from the list. The next player and all other players in the chain perform the same computation until the list contains no players. At the end, $E(x)$ and $E(y)$ are sent to P_m.
4. Player P_m computes:

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

   where α_1, α_2, β_1, and β_2 are randoms less than $MAXINT$. Player P_m 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_m.
6. Player P_m computes the ratio $\frac{x}{y}$ as $(\beta_1 - \delta \cdot \beta_2)/(\delta \cdot \alpha_2 - \alpha_1)$ and forwards the answer to all other players.

Analysis This protocol is the natural choice when the number of players is 2, because the previous division protocols apply constraints to the smallest number of players. This protocol, however, 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 should be carefully constructed to minimize the probability of success. Instead of using large prime β_1 and β_2 or values composed of a few prime numbers, we compose the coefficients as a product of a possibly large number of integers in the range of expected y. In other words, we compute $\beta_1 = \prod_{i=1}^{k} \hat{y}_i$ for some k such that $\beta_1 < MAXINT$ and where each $\hat{y}_i$ is close to the expected value of y and is possibly present in both β_1 and β_2. In this case, even if player P_1 is successful in factoring the numbers, he will not be able to determine what the actual value of y is.

When the number of players is larger than 2, correctness of the protocol depends on two players. If those two players collude, they can obtain x and y (but not individual $x^{(j)}$ or $y^{(j)}$'s). Individual values $x^{(j)}$ and $y^{(j)}$ can also be revealed if the player who has the decryption key and two players in the chain before and after P_j collude. The random nature of routing at step (3) and assignment of roles at step (1) significantly lowers the probability of revealing individual inputs, but is still possible when the number of players who may collude is large. Also, the player who possesses the decryption key has more power than any other player.

Therefore this protocol should be used when $m = 2$ or when the probability of collusion by many player is low.

The protocol consists of two rounds, during which all players exchange $O(m)$ messages. Each player needs only to perform two encryptions in modular arithmetics. Player P_1 additionally creates a key pair and performs two decrypt operations. Player P_m also performs a small (constant) number of multiplication and exponentiation operations in modular arithmetics.

Lastly, we give a protocol that is secure against collusions of up to $m-1$ players and no partial information about x and y is leaked. 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 without External Parties 4*

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

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

Interaction The protocol is carried out as described below:

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 $q_j = E_j(y^{(j)})$.
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 $j \neq i$). 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 randoms (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:
   $$p_j = p_j^{\alpha_i} \cdot E_j(a_{i,j}) = E_j(x^{(j)}\alpha_i) \cdot E_j(a_{i,j}) = E_j(x^{(j)}\alpha_i + a_{i,j})$$
   $$q_j = q_j^{\beta_i} \cdot E_j(b_{i,j}) = E_j(y^{(j)}\beta_i) \cdot E_j(b_{i,j}) = E_j(y^{(j)}\ beta_i + b_{i,j}).$$
   (b) Player P_i now updates his own p_i and q_i by doing
   $$p_i = E_i(\alpha_i \cdot D_i(p_i) - \textstyle\sum_{j \neq i} a_{i,j})$$
   $$q_i = E_i(\beta_i \cdot D_i(q_i) - \textstyle\sum_{j \neq i} b_{i,j}).$$
   Note that the above decryption and re-encryption of p_i are not necessary and the computation could have been performed on encrypted items, but we chose to do the arithmetic on unencrypted 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}. If $i = m$ then 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 $y^{(j)} = D_j(q_j)$.

   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 (and no one will ever 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 $\beta y + r_y$, where r_x and r_y are negligible compared to αx and βy (more discussion of this follows).
4. A secure summation protocol (see section 4.3) computes (and lets everyone know) the sum of the $y^{(j)}$'s. That is, everyone now knows $\beta y + r_y$, and each player P_j computes $\delta^{(j)} = x^{(j)}/(\beta y + r_y)$.
5. Every player P_j reveals to all the others the ratio $t_j = \beta_j/\alpha_j$.
6. Every player P_j computes $\delta^{(j)} t_1 t_2 \cdots t_m$, which results in the approximation of $\frac{x}{y}$ being split among m players. To recover the value, they run a secure summation protocol, and each player learns $\frac{x}{y}$ with the precision required.

Analysis The aggregate randoms 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:

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

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

which converges to $\alpha x/\beta y$ when $r_x \ll \alpha x$, $r_y \ll \beta y$, and r_x and r_y are random. 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 $MAXINT^{m-1}$.

To further minimize the probability of successful computation of x or y, we might decide to compose single α_i's and β_i's from a set of factors close in the range to the expected values of x and y, respectively, similar to the technique described in analysis of protocol 5.

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. It is conducted in 2 rounds, with the total communication of $O(m^2)$ items (or $O(m)$ messages). Computational complexity at each player is bounded by key creation and $O(m)$ encryptions.

4.3 Secure summation protocols

In this section, we present two secure summation protocols. Assuming that each player has his own private input, the goal of the summation protocols is to find the sum of the players' input data. We first present a protocol that works in the presence of semi-honest players, then we give a robust but more expensive protocol that works even when players may collude.

Protocol 7 *Secure Summation Protocol 1*

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

Output Player P_j, $1 \leq j \leq m$, learns x.

Interaction This protocol has following steps:

1. All players engage in Protocol 1 with security parameter $k = 1$. Each player P_j provides $x^{(j)}$ as his input and stores the output as $y^{(j)}$.
2. All players jointly and randomly select a player who will begin the summation. Without loss of generality, assume that player P_1 is selected.
3. Player P_1 creates a list of all players, crosses out his name from the list and sends $y^{(1)}$ to a randomly chosen player P_i from the list; assume player P_2 is chosen. P_2 crosses out his name from the list, computes a sum of the received and his own values $y^{(1)} + y^{(2)}$, and sends it to another randomly chosen player from the list. The protocol proceeds in this fashion until every player P_j participates and sends the result of its computation to another available player. At the end, the last player on the chain distributes the result of the m-way summation y to all players.

Analysis This protocols exhibits information asymmetry with respect to different players, i.e., the first player sends his value of $y^{(1)}$, while in all subsequent computations $y^{(j)}$ is always a part of the value being sent. This means that in order to retrieve more information about player P_1's input data, a collusion of the next player on the chain and the player with whom P_1 shares his input is sufficient. Contrast this with learning about input of another player in the chain where collusion of a large number of players is required. Consequently, this model should be used where we can assume primarily non-colluding players. Randomness of the protocol, however, makes collusions probabilistically more difficult to succeed because it is not known in advance with whom players are going to share their inputs and what the summation chain will be.

This protocol proceeds in two rounds, with total communication of $O(m)$ messages. Computational complexity at each player is $O(1)$.

Next, we provide a more robust version of this protocol. No information about the individual inputs can be learned as long as the number of colluding players is less than $m - 1$.

Protocol 8 *Secure Summation Protocol 2*

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

Output Player P_j, $1 \leq j \leq m$, learns x.

Interaction This protocol has the following steps:

1. All players engage in Protocol 1 with security parameter $k = m - 1$. Each player P_j provides $x^{(j)}$ as his input, stores the result as $y^{(j)}$, and then sends $y^{(j)}$ to all other players.
2. Upon receiving the values of $y^{(j)}$ from everyone, each player computes $x = \sum_{j=1}^{m} y^{(j)}$.

Analysis This protocol is resistant to colluding behavior, as long as the number of colluding players is less than $m - 1$. The protocol requires two rounds of communication and the total of $O(m^2)$ message exchanges. Its computational complexity at each player is $O(m)$.

4.4 Secure comparison protocol

This section contains description of a secure comparison protocol, the goal of which is to compare two split items in a secure way without revealing any information about the values being compared. The idea behind our protocol is to convert an m-party comparison problem to a two-party comparison problem. Two-party comparison can be solved using Yao's millionaire protocol [23].

Protocol 9 *Secure Comparison Protocol*

Input Player P_j, $1 \leq j \leq m$, supplies two values $x^{(j)}$ and $y^{(j)}$.

Output Player P_j, $1 \leq j \leq m$, receives true if $x \geq y$, and false otherwise.

Interaction This protocol has the following steps:

1. All players randomly and jointly select two among of them to conduct the Yao's comparison. Without loss of generality, assume that players P_1 and P_2 are selected.
2. Player P_j, $3 \leq j \leq m$, generates a random number $a^{(j)}$.
3. Players P_3 through P_m engage in a secure split protocol twice, providing their values $a^{(j)}$ as input. At the end of the protocols, player P_j holds two new values $r_x^{(j)}$ and $r_y^{(j)}$, such that $\sum_{j=3}^{m} r_x^{(j)} = \sum_{j=3}^{m} r_y^{(j)} = \sum_{j=3}^{m} a^{(j)}$.
4. Player P_j, $3 \leq j \leq m$, sends $x^{(j)} + r_x^{(j)}$ to player P_1, and $y^{(j)} + r_y^{(j)}$ to player P_2.
5. Player P_1 sets $x_1 = x^{(1)} + x^{(3)} + r_x^{(3)} + x^{(4)} + r_x^{(4)} + \ldots + x^{(m)} + r_x^{(m)}$ and $y_1 = y^{(1)}$. Player P_2 sets $x_2 = x^{(2)}$ and $y_2 = y^{(2)} + y^{(3)} + r_y^{(3)} + y^{(4)} + r_y^{(4)} + \ldots + y^{(m)} + r_y^{(m)}$.
6. Players P_1 and P_2 engage in Yao's protocol to jointly and securely compare values $(x_1 - y_1)$ and $(y_2 - x_2)$ (which works since $x_1 + x_2 \geq y_1 + y_2 \iff x_1 - y_1 \geq y_2 - x_2$).
7. Players P_1 and P_2 distribute the result of the computation to players P_3 through P_m.

Analysis Even though in this protocol we add random numbers to the actual input data, it works because the same number is added to both the sum of $x^{(j)}$'s and $y^{(j)}$'s. This protocol is also secure against colluding players with respect to revealing individual shares, as long as the number of colluding players is less than $m - 1$. It leaks partial information if the players who conduct the comparison collude: those players can discover the difference between x and y but no other information.

Yao's comparison protocol can be efficiently performed in a constant number of rounds and a linear (in the number of bits of the compared numbers) amount of communication and local computation using circuit simulation: first create a circuit for comparison and then use the constant-round circuit simulation technique first described in [22] and further reviewed and developed in [12]. Consequently, the overall protocol is conducted in $O(1)$ rounds, with the total of $O(m)$ message exchanges. Computational complexity at each player who does not conduct the comparison is $O(1)$, and $O(m)$ otherwise.

5 Secure Time-Series Forecasting

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

5.1 Moving average

In the case of moving average forecasts, the goal is to find the behavior of the function at time $t+1$ relative to the current time t. The value can be computed as in the following:

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

We provide two approaches to the moving average problem. One is to use a secure division protocol, and another is to use binary search in combination with the secure comparison protocol, both of which we present next.

Protocol 10 *Secure Moving Average Protocol Using Division*

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.

Interaction The moving average forecast can be performed easily if we utilize one of the division protocols. The protocol steps are then as the following:

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 depends on the underlying secure division protocol. Communication and complexity requirements are outweighted by any division protocol because the first step of this protocol does not involve communication and adds $O(1)$ computation (n is constant).

We now present an alternative solution. Due to difficulties of handling floating point numbers, let us assume that we are going to compute only a fixed number k of digits of x after the point. In other words, we can compute $x' = 10^k x$, where x' is integer and k is a non-negative integer that specifies precision of the calculation. Using the same notation as above, we have:

$$\sum_{j=1}^{m} 10^k \big(d_{t-n+1}^{(j)} + \ldots + d_{t-1}^{(j)} - (n-1)d_t^{(j)}\big) = x' \sum_{j=1}^{m} nd_t^{(j)} \tag{3}$$

Here we can use a binary search on x' to find its value in such a way that all players jointly compute the value without revealing their individual inputs.

Protocol 11 *Secure Moving Average Protocol Using Binary Search*

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 $\frac{F_t - d_t}{d_t}$.

Interaction This protocol proceeds in the following steps:

1. Each player P_j computes $a^{(j)} = 10^k \big(d_{t-n+1}^{(j)} + \ldots + d_{t-1}^{(j)} - (n-1)d_t^{(j)}\big)$ and $b^{(j)} = nd_t^{(j)}$.

2. All players agree on the value of x' (which initially could be 0, the middle of the interval $[-10^k, 10^k]$).
3. All players conduct a comparison protocol (protocol 9), where each player P_j supplies $a^{(j)}$ and $x'b^{(j)}$. If the result is true, they agree on a larger x'; if the result is false, they pick a smaller value of x'.
4. Repeat steps (2)–(3) until x' is determined.

Analysis Using binary search, this protocol results in at most $O(\log(2 \cdot 10^k)) = O(1)$ rounds. The overall communication complexity for all players and computational complexity at each player is the same as for the comparison protocol, since the comparison protocol introduces most complexity and is invoked a constant number of times. This protocol is also as resistant to colluding behavior as the underlying comparison protocol. We can make it to be more robust if the players who perform comparison are chosen randomly in every round. This strategy will minimize the amount of information that attackers might learn.

As an alternative, in order to decrease the number of rounds of the protocol, one might consider a model with only one round but higher communication and computational costs. In this case, each player computes his values and follows the protocol for all possible values of x', and by doing this determines the closest value of x' to the true one.

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) in this case becomes:

$$x = \frac{F_t - d_t}{d_t} = \frac{\left(\sum_{i=0}^{n-1} w_i d_{t-i}\right) - d_t}{d_t} = \frac{w_0 d_{t-n+1} + \ldots + w_{n-2} d_{t-1} - (1 - w_{n-1}) d_t}{d_t}$$

Similarly to the previous case, we provide two different algorithms for computing the value: one which uses division, and another that uses binary search.

Protocol 12 *Secure Weighted Moving Average Protocol Using Division*

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.

Interaction The steps of the protocol are as below:

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 10.

For the binary search protocol, equation (3) can be re-written in this case as:

$$\sum_{j=1}^{m} 10^k \left(w_0 d_{t-n+1}^{(j)} + \ldots + w_{n-2} d_{t-1}^{(j)} - (1 - w_{n-1}) d_t^{(j)}\right) = x' \sum_{j=1}^{m} d_t^{(j)}$$

The protocol for computing the weighted moving average using binary search is also similar to the same protocol for computing the moving average.

Protocol 13 *Secure Weighted Moving Average Protocol Using Binary Search*

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.

Interaction The protocol is performed as follows:

1. Each player P_j computes $a^{(j)} = 10^k \left(w_0 d_{t-n+1}^{(j)} + \ldots + w_{n-2} d_{t-1}^{(j)} - (1 - w_{n-1}) d_t^{(j)}\right)$ and $b^{(j)} = d_t^{(j)}$.
2. Steps (2)–(4) are the same as in protocol 11.

Analysis See analysis of protocol 11.

5.3 Exponential Smoothing

To simplify joint computation of exponential smoothing, we rewrite the formula as:

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

Assume α is public, F_{t-1} is calculated during the previous execution of the protocol, and is additively split between m players.

Protocol 14 *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.

Interaction The protocol can be conducted through the following 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 more precise 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 values, can also 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, then they can solve it in one of the two ways:

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

(b) They can omit the step of the protocol where the value of y's is computed using the division protocol and use their original values of y'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 some additional information about other players' input that should be kept secret. Changing the protocols 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 in terms of both computational power 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 have the value of zero, then the execution is suspended. Each player will be notified that this computation cannot be carried out, and they have two options: they can either abort the protocol or continue its execution, but in which case information about the missing values will be revealed to all players. If all of the players decide to continue, that 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 can be rewritten as:

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

where the values of 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 the value of B can be computed only after a is known as a result of joint computation).

Protocol 15 *Secure Linear Regression Protocol*

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

Output Player P_j, $1 \leq j \leq m$, learns the coefficients a and b such that the deviation of the points in $\vec{y} = a\vec{x} + b$ is minimized.

Interaction The protocol proceeds in the following steps:

1. All players engage in a secure division protocol n times to compute the values of y_i for $1 \leq i \leq n$, where the y_i values remain additively split among all players.
2. Each player P_j locally computes $a^{(j)} = A\Big(n \sum_{i=1}^{n} x_i y_i^{(j)} - \big(\sum_{i=1}^{n} x_i\big) \sum_{i=1}^{n} y_i^{(j)}\Big)$.
3. All players engage in a secure summation protocol to compute $a = \sum_{j=1}^{m} a^{(j)}$.
4. Each player P_j locally computes $b^{(j)} = \sum_{i=1}^{n} (y_i^{(j)}/n)$.
5. All players engage in a secure summation protocol to compute $b = \sum_{j=1}^{m} b^{(j)} - B$, where the sum can be computed through the protocol itself but the value of B can be later subtracted by each player.

Analysis This protocol is as secure against colluding players as its underlying blocks are (namely, the division and summation protocols). Communication and computational complexity of the protocol is the higher of the division and summation protocols used because both of them are invoked a constant number of times (the number of points n is constant).

7 Conclusions and Future Work

In this work, we provide privacy-preserving solutions to collaborative forecasting and benchmarking, which can be used to increase reliability of local forecasts, perform evaluation of the local and global trends, and obtain more precise correlation of data that might be essential to the business. We provide both building blocks and fully developed protocols to perform a number of different forecasting methods based on time-series as well as regression techniques.

The building blocks are general enough to be used in other protocols for forecasting and benchmarking, as well as 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 perform computation on floating point numbers.

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

- The number of time-series forecasting methods covered can be extended to other forecasting techniques.
- Among with providing short-range forecasting, we would like to be able to perform long-range forecasts. Long-range forecasting takes into account seasonal changes and other long-term patterns.
- We also would like to design protocols to cover other types of regression techniques for benchmarking collaboration. This will allow us to draw reliable conclusions for different types of data distributions.
- A number of protocols provided in this paper can be made more robust against other types of malicious behavior.

References

[1] K. Allan, M. Stemper, and O. Tucker. "Collaborative Forecasting," available at *http://e-business.pwcglobal.com/pdf/CollaborativeForecasting.pdf*

[2] M. Atallah and W. Du. "Secure multi-party computational geometry," in *WADS2001: 7th International Workshop on Algorithms and Data Structures*, pp. 165–179, 2001.

[3] M. Atallah, H. Elmongui, V. Deshpande, and L. Schwarz. "Secure Supply-Chain Protocols," In *IEEE International Conference on Electronic Commerce*, pp. 293-302, 2003.

[4] O. Baudron and J. Stern. "Non-interactive Private Auctions," in *Financial Crypto'01*, Springer-Verlag, 2001.

[5] Collaborative Planning, Forecasting, and Replenishment (CPFR), *http://www.cpfr.org/Members.html*.

[6] W. Du. "A Study of Several Specific Secure Two-party Computation Problems," *PhD thesis*, Purdue University, West Lafayette, Indiana, 2001.

[7] W. Du and M. Atallah. "Privacy-preserving cooperative scientific computations," in *14th IEEE Computer Security Foundations Workshop*, pp. 273–282, 2001.

[8] W. Du and M. Atallah. "Privacy-preserving statistical analysis," in *Proceedings of the 17th Annual Computer Security Applications Conference*, pp. 102–110, 2001.

[9] J. Evans. "Applied production and operations management" (4th Edition), *West Publishing Company*, 1993.

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

[11] E. Frazelle. "Supply chain strategy: the logistics of supply chain management," *McGraw-Hill*, 2002.

[12] O. Goldreich. "Cryptography and cryptographic protocols," *Distributed Computing*, 16(2–3), pp. 177–199, 2003.

[13] O. Goldreich. "Secure Multi-party Computation" (working draft), available at *http://www.wisdom.weizmann.ac.il/home/oded/public_html/pp.html* (2001).

[14] O. Goldreich, S. Micali, and A. Wigderson. "How to play any mental game," in *Proceedings of the 19th Annual ACM Symposium on Theory of Computing*, pp. 218–229, 1987.

[15] S. Goldwasser. "Multi-party computations: Past and present," in *Proceedings of the 16th Annual ACM Symposium on Principles of Distributed Computing*, Santa Barbara, CA USA, August 21-24, 1997.

[16] John Galt Solution, Inc. *http://www.johngalt.com/*

[17] Y. Lindell and B. Pinkas. "Privacy preserving data mining," in *Advances in Cryptology - CRYPTO 2000*, pp. 36–54, 2000.

[18] T. Rabin and M. Ben-Or. "Verifiable secret sharing and multiparty protocols with honest majority" (extended abstract), in *Proceedings of the Twenty First Annual ACM Symposium on Theory of Computing*, pp. 73–85, 1989.

[19] B. Schneier. "Applied cryptography: protocols, algorithms, and source code in C" (2nd Edition), *John Wiley & Sons, Inc.*, 1995.

[20] H. Singh. "Collaborative Forecasting," available at *http://www.supplychain.com/docs/collaborativeforecasting.pdf*, 2002.

[21] W. Stevenson. "Production/Operations Management" (4th edition), *Richard D. Irwin, Inc.*, 1993.

[22] A.C. Yao. "How to generate and exchange secrets," in *Proceedings of the 27th Annual IEEE Symposium on Foundations of Computer Science*, pp. 162–167, 1986.

[23] A. Yao. "Protocols for secure computations," in *Proceedings of the 23rd Annual IEEE Symposium on Foundations of Computer Science*, 1982.