Incentivizing MPC

Kyle Hogan, Aanchal Malhotra, Sarah Scheffler

January 3, 2017

1 Introduction

Secure Multiparty Computation (MPC) allows mutually distrusting parties to learn the output of a function computed on their private inputs without the use of a trusted party or forcing any party to reveal their private data. However, MPC computations are often expensive - parties need to exchange many rounds of communication and must remain online for the duration of the computation, and they may also spend significant person-time organizing the MPC with the other organizations involved. Also, in competitive settings, it is not clear whether the benefits gained from learning the output of the computation outweigh the indirect losses incurred by helping their competitors learn their own output values. In many cases, some potential contributing parties feel that the benefit of learning the output does not adequately compensate them for the cost of performing the computation itself. This is particularly true for those parties who are already in the dominant position, as they tend to have the greatest amount or quality of data. In a sense, they will contribute the most to the shared computation, but they gain the least benefit relative to their knowledge prior to learning the output of mutual computation.

In this work, we address this issue by designing a modular framework on how to assign *value* to each party's input data and how to use this assignment to correspondingly adjust the *utility* of their output, thus incentivizing them to participate in the computation.

We provide methods of increasing the utility of computation outputs such as returning the results to parties with higher value data first, or giving parties with higher value data a less noisy result. We believe that this will encourage participation in the computation by parties with more valuable datasets that would otherwise have not benefited enough to offset their cost and competitive edge, thus increasing the utility for all parties participating.

We also investigate the use of this framework in the specific use case of evaluating datasets for purchase, by directly returning the value computations themselves.

Section 2 briefly describes prerequisites to this work, and frameworks used. In section 3, we describe an overview of our modular framework. In section 4, we give a more in-depth description of each phase of our framework, with examples of each step and a discussion of how to make the different phases work with each other best. Finally, in section 5, we fully describe two example entire paths through our framework. One example features a pseudocode implementation.

2 Prerequisites

Secure Multiparty Computation Secure Multiparty Computation (often abbreviated simply MPC) is a cryptographic technique for having multiple parties conduct a shared function evaluation on secret inputs. Each party will learn their output of the computation, but will learn nothing about another party's input that could not have been learned from their input and the output otherwise. We assume a working knowledge of MPC techniques in the other sections of this document.

Secret Sharing MPC achieves this using secret sharing [GMW87, Yao86, BOGW88]. Parties will split their input into 'shares' such that fewer than a threshold number of shares will reveal nothing about the original input. Parties can then send one share of their secret input to every other party and computation can be performed over these shares rather than the inputs themselves. At the end of the computation each party will hold a share of the result and at least a threshold number of them can recombine their shares to obtain the result.

SPDZ Framework Since MPC was first introduced, several frameworks implementing MPC have been developed including Sepia[BSMD10], Sharemind[BLW08], VIFF[DGKN09], and SPDZ[DKL+13]. Ultimately we selected SPDZ, the successor to VIFF, for out implementations, but the protocols discussed are independent of the underlying framework and could be implemented on others as well.

3 Modular Framework

Our framework consists of two additional phases to be added to a normal MPC shared function evaluation, one before and one after the computation phase.

The workflow for a given computation proceeds as follows:

1. **Objective Function Calculation** In our setting, the objective function indicates how much (based on some metric) each party's data contributes to the result of the computation. For this, all inputs are secret shared among the parties. According to some user-defined objective function, some function of one party's dataset is calculated as a function of every party's dataset. It may be possible to calculate more than one party's objective function at once, or it may be necessary to do multiple rounds so that all parties' objective functions can be calculated.

2. **Computation** In this step, the original shared computation is performed. We assume that the output can be altered with minimal interference to the computation itself.

3. **Output Alteration** Based on the objective function, we alter the results of the output. This could involve returning results to parties in a certain order, changing the accuracy of the results returned to each party, or even having some parties learn the output of the objective function itself.

Each of these steps is discussed in more detail below.

3.1 Objective Function Calculation

The choice of objective function is at the heart of this framework. The purpose of the objective function is for a party to assign a definition of "value." The goal of the objective function is effectively to be a transformation from what a party wants in a dataset to an output where a dataset with a higher objective function is "better."

The problem of evaluating the value of a dataset has been a very large field of research for many years now. We do not want to be in the business of defining some global definition of value for datasets. There is no such thing as an exhaustive list of objective functions, and any attempt we might make to pick the "best" objective functions would be pointless, since each organization is going to be operating on different data, and desires different qualities in their data. Instead, in section 4.2 we have: (1) implemented a few very basic objective functions, mostly for the purpose of testing our framework, (2) listed some objective functions that we think are specifically well-suited for MPC, especially for parties that would be motivated to do MPC only with the added incentive provided by our framework, and (3) shown how the objective function can be used in many different ways in our examples.

3.2 Computation

The computation is almost exactly the same as whatever computation the parties were going to perform without the output alteration mechanism. This is completely user-defined; we make no attempt here to provide sample computations other than our examples. However, we do point out that the computation should have certain traits for use in this framework. The output of the objective function could affect both the input and output to the computation. However, the intended use of this framework is for the objective function output to affect *only* the output of the computation, in the output alteration phase. Altering the input to the computation is allowed, but it does somewhat break modularity.

Here we only consider computations of a single round. Computations that involve multiple rounds of output or asynchronous output *could* still work with this framework, each round with

its own output alteration phase, but here we limit our attention to a single round of output for simplicity's sake.

It may be that the actual purpose of this MPC was to learn some function of the objective functions themselves, rather than to perform an arbitrary computation. In this case, we consider the computation to be "null," in the sense that it has no inputs or outputs.

3.3 Output Alteration

In the output alteration phase, we take the output of the computation, and modify it based on the results of the objective function. There are several potential methods for this. We consider three:

1. Using timed-delay MPC to affect the order of MPC results (see section 4.2)
2. Altering the accuracy or fidelity of results (see section 5.1)
3. Returning the values or rankings of the objective function directly (see section 4.3 and 5.2)

4 Choosing an objective function, computation, and output alteration

Each phase of the computation can be considered on its own, and we do our best to maintain a modular design where many objective functions, computations, and output alterations can be combined without too much effort on the user's part. However, there is also a lot of benefit to designing objective functions, computations, and output alterations that play well with each other. In this section, we discuss several objective functions and ideas for corresponding output alterations.

4.1 Objective function choices

Meandiff The mean difference, which we abbreviated to *meandiff*, is a very simple statistical test meant to determine how much each party's dataset differs from the rest of the parties' combined datasets. If μ_{all} is the mean over all parties' datasets, and μ_i is the mean of all parties' datasets *except* party i, then the meandiff is

$$|\mu_{all} - \mu_i|.$$

This can be calculated for each party without revealing μ_all or μ_i. We simply calculate each in MPC using each party's sum of all their data points and number of all their data points, and calculate the difference in MPC. If there are N parties, labeled 1 through N, and party j has input s_j and n_j, the sum of their data points and number of their data points respectively, then the meandiff for party i is:

$$\mathsf{meandiff}_i = \mu_{all} - \mu_i = \left| \frac{\sum_{j=0}^{N} s_j}{\sum_{j=0}^{N} n_j} - \frac{\sum_{j=0, j\neq i}^{N} s_j}{\sum_{j=0, j\neq i}^{N} n_j} \right|$$

To evaluate the viability of our model, we implemented a simple example program to calculate meandiff[HMS16]. Note that due to technical issues with the SPDZ implementation[KRS16] involving the inability to reveal a secret floating point number, as well as complications with the input/output method provided by SPDZ, we did not finish an implementation on floats in the time allotted.

Statistical Tests A statistical test provides a mechanism for making quantitative decisions. If we have data that allows us to determine the distribution of output of the computation before and after the addition of a particular party's data, we can reliably determine if their data have impacted the output. Meandiff is one example of this. We did not implement any of these beyond meandiff, but [Kam15] provides an entire statistical analysis suite for MPC. Her implementation is in Sharemind, but the algorithms could be adapted for use in SPDZ as well.

Outlier Detection Parties may have incomplete representations of global data from their own local datasets. In particular, they could have a data point or cluster of points that appear to be outliers in relation to the rest of their local points, but are actually well within the reasonable expectations for global points. In this case it could be incredibly valuable for a party to obtain data that helped make sense of what they had locally.

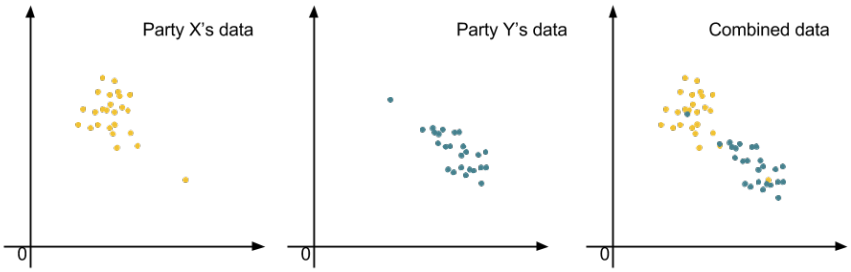

Figure 1: Both parties would benefit from learning the other's data

Using outlier detection as the objective function would return a different ranking to each party showing how much that party valued each other party's data. A dataset similar to the party's own data would receive low ranking as would one that overlapped with none of their points. The highest ranked datasets would be ones that intersected with some of the party's data, showing evidence of a global trend that had not been apparent from the local data alone.

4.2 Output alteration choices

Timed-Delay MPC In an idea proposed by [AGP16], the delivery of results in an MPC can occur in a specific order, with a delay between each party's output. In *timed-delay MPC*, a permutation on the parties is used as extra input to an MPC, and this permutation is used to specify an order in which the results will be output to each party. There is a specified delay between each of these outputs, giving the party who got the output first more time to take advantage of the information that only they have.

This fits very well into our framework - we choose an objective function that has a permutation as its output, then we use timed-delay MPC as our output alteration step. The results of the MPC are then delivered to the parties in an order determined by the agreed-upon objective function.

4.3 Dataset Purchasing

One direct way to use the information provided by an objective function is to determine how valuable a dataset is for purchase. In this setup, you have a number of buying parties and a number of selling parties (the buyers and sellers may overlap). A buyer has an objective function by which they evaluate potential datasets they would like to buy. The output of the objective function on each individual dataset may be sensitive, but since we have many sellers, we can return to buyers the *ranking* of the sellers' databases rather than the value of the objective function itself. (Of course, the sellers must agree that this ranking is not itself sensitive.) In this case, the computation is ignored entirely, and the output alteration takes the objective function's evaluation of each seller's database, ranks them, and returns the ranking to the buyer.

Many variations on this scheme are possible, each with its own interesting results. If all buyers agree on a single objective function (though their inputs will still be different), then the sellers could find the buyer that wants their database the most, or even learn how much each buyer wants their database according to that objective function. The sellers could also use this objective function to see how they rank in comparison with the other sellers.

The use case of using an objective function to evaluate datasets in preparation for buying and selling data is a topic worthy of future investigation. The following questions are ones that our framework is capable of answering in its current form, though these questions would probably better fit into a framework that does not assume it is evaluating a generic computation the way ours does.

Questions for data buyers:

1. "Which seller's data is the most valuable to me?"
2. "How valuable is each seller's data to me?"
3. "How similar are my data preferences to other buyers?"
4. "Is there a certain seller whose data I want more than any other buyer?"

Questions for data sellers:

1. "Which buyer wants my data the most?"
2. "How much does each buyer want my data?"
3. "How valuable is my dataset compared to other sellers?"
4. "Is there a certain buyer that wants my data more than any other buyer?"
5. "Is there a certain buyer that wants my data above all other sellers' data?"

Future work on this project will likely involve focusing on some of these questions, or expanding them to the scenario of a "data auction."

5 Example Workflows

Two examples through our entire framework are listed here. One goes through all the steps as a generic example of our framework, and the other focuses on the dataset selling scenario, where we only care about the outputs of the objective function and we skip the computation step entirely.

5.1 Example 1: Modifying calculated map routes according to driver recklessness

In this example, we examine the situation where a collection of drivers want to determine the best route for each of them to take to get to their destinations. It is beneficial for all the drivers if they are sent on different routes, even if some of these may be longer, to reduce the effect of traffic on their overall driving times.

However, no single party wants to be sent on a longer route so in order to determine which parties receive which navigational directions they will first perform a computation to rank the drivers in terms of safety. Ranking them this way serves a twofold purpose: it promotes safe driving and, as the fastest routes though suburbs can often be on neighborhood streets rather than major throughways, it routes unsafe drivers away from the areas where they would be most likely to injure pedestrians.

To determine which drivers are safest each driver will contribute data from their phone on acceleration and deceleration habits. Drivers who often accelerate rapidly or stop suddenly will be given a lower ranking. Additionally, drivers who are often involved in accidents will be ranked lower.

This ranking will serve as input to the navigation computation along with the drivers start and end locations. The computation will globally optimize routes for minimal overall traffic by giving lower ranked drivers the less preferred routes. Some additional care should be taken to ensure that these longer routes do not also go through a neighborhood.

Once both the objective function ranking the drivers and the computation determining their routes are complete, each driver will be returned directions to their destination.

5.2 Example 2: Buying traffic data for a specific (secret) region

In this example, we examine the use of our framework for evaluating datasets for purchase. Recall that we discussed this idea earlier, and observed that there are many different objectives that buyers and sellers of data could use to facilitate the purchasing of data. Here, we examine only one of these many possibilities.

Suppose several data buyers are interested in the traffic behavior of certain secret regions in Boston. Several different sellers of traffic behavior exist (different map mobile applications, for

instance), but their proprietary data is valuable and they do not want to release information about how many data points they have for different streets, unless it will help them sell their dataset to a buyer.

In this case, each buyer will use the same objective function evaluated on all the sellers, but with different inputs. Buyers will enumerate streets that they are interested in, and input these to the objective function. The objective function will determine a ranking, for each buyer, of which seller contains the largest number of data points on streets they are interested in. If the sellers were concerned that a buyer might game the system by asking for too many streets, the objective function could impose a limit on how many streets the buyer can ask about.

Each buyer will learn a ranking of sellers that tells them which seller's data is the most valuable to them, and the sellers will learn nothing. (Keep in mind that we could have done a more involved computation and had the buyers learn the number of data points desired, or had the sellers learn a ranking of buyers were interested in them, or any number of other computations.)

5.2.1 Pseudocode Implementation

We do not have a completed implementation for this example, but we do have a pseudocode description of how it would work, and a dataset of Boston traffic jams collected by the traffic mobile app Waze[Sav15] that would serve as real-world example data. The data were to be distributed unevenly among several different sellers, and the buyers would be randomly assigned streets about which they want to buy data.

Say there are S sellers and B buyers. They follow the following steps to do this computation:

1. All parties use a public encoding of street names to integers. This could be accomplished by using a shared map of Boston, enumerating all street names, and assigning each a number. Let the numbers range from 0 to N.

2. Each seller locally calculates a length-N array where the value at a certain index is equal to the number of data points the seller has at the street corresponding to the index in the array.

3. Each buyer locally creates a list of the street integers they are interested in.

4. Beginning the SPDZ protocol, the secret seller arrays and secret buyer streets are secret shared across all parties.

5. For each seller array s from 1 through S, we create an accumulator for each buyer b from 1 through B and then loop over the seller's array from index $i = 0$ to N. At each i, we set $acc_{b,s} = acc_{b,s} + s[i] * x_b$, where x_b is a disjunction of all the streets desired by the buyer with accumulator b. (For example, if buyer b wants data about streets 1 and 5, then $x_b = (i = 1 \| i = 5)$.) All of these steps are done on secret shares. The accumulator $acc_{b,s}$ will get the total number of data points possessed by s that are desired by b.

6. Using comparisons of integers in MPC, use the $acc_{b,s}$ values to compute a ranking of the accumulators for a given buyer.

7. Reveal these rankings of sellers (not the accumulators themselves) to the buyer they correspond to.

6 Future Work

We started this project investigating ways in which we could increase the incentives for parties who already have good data to participate in an MPC that would normally disproportionately help their competitors. In the process of doing this work, however, we moved to a slightly different question - how can MPC be used among sellers and buyers of data to help them buy and sell their data optimally? We are especially interested in doing more research into how these concepts could be applied to an *auction* of data happening in MPC.

We would also like to investigate other ways in which MPC could be brought to more users. At present, the barriers to entry are still very high, both in terms of libraries of code and in terms of communicating the right level of understanding to the right people. If we wish this framework to be used in any practical capacity, good communication and usability are essential.

References

[AGP16] Pablo Daniel Azar, Shafi Goldwasser, and Sunoo Park. How to incentivize data-driven collaboration among competing parties. In *Proceedings of the 2016 ACM Conference on Innovations in Theoretical Computer Science*, ITCS '16, pages 213–225, New York, NY, USA, 2016. ACM.

[BLW08] Dan Bogdanov, Sven Laur, and Jan Willemson. *Sharemind: A Framework for Fast Privacy-Preserving Computations*, pages 192–206. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008.

[BOGW88] Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation. In *Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing*, STOC '88, pages 1–10, New York, NY, USA, 1988. ACM.

[BSMD10] Martin Burkhart, Mario Strasser, Dilip Many, and Xenofontas Dimitropoulos. Sepia: Privacy-preserving aggregation of multi-domain network events and statistics. In *Proceedings of the 19th USENIX Conference on Security*, USENIX Security'10, pages 15–15, Berkeley, CA, USA, 2010. USENIX Association.

[DGKN09] Ivan Damgård, Martin Geisler, Mikkel Krøigaard, and Jesper Buus Nielsen. *Asynchronous Multiparty Computation: Theory and Implementation*, pages 160–179. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009.

[DKL+13] Ivan Damgård, Marcel Keller, Enrique Larraia, Valerio Pastro, Peter Scholl, and Nigel P. Smart. *Practical Covertly Secure MPC for Dishonest Majority – Or: Breaking the SPDZ Limits*, pages 1–18. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013.

[GMW87] O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game. In *Proceedings of the Nineteenth Annual ACM Symposium on Theory of Computing*, STOC '87, pages 218–229, New York, NY, USA, 1987. ACM.

[HMS16] Kyle Hogan, Aanchal Malhotra, and Sarah Scheffler. incentivizing-mpc. https://github.com/sarahscheffler/incentivizing-mpc, 2016.

[Kam15] Liina Kamm. *Privacy-preserving statistical analysis using secure multi-party computation*. PhD thesis, University of Tartu, January 2015.

[KRS16] Marcel Keller, Dragos Rotaru, and Nigel P. Smart. Spdz-2. https://github.com/bristolcrypto/SPDZ-2, 2016.

[Sav15] Curt Savoie. Waze jam data. https://data.cityofboston.gov/Transportation/Waze-Jam-Data/yqgx-2ktq, 2015.

[Yao86] Andrew Chi-Chih Yao. How to generate and exchange secrets. In *Proceedings of the 27th Annual Symposium on Foundations of Computer Science*, SFCS '86, pages 162–167, Washington, DC, USA, 1986. IEEE Computer Society.