MPC for Secure Data Analysis in Higher Education

Kevin Dong Adam Doyle Andre Rosado Swagat Adhikary Prajwal Moharana

***Abstract*—This project implements Multi-Party Computation (MPC) techniques to securely gather and analyze sensitive data about the student body at the University of North Carolina without compromising individual privacy. Using a combination of frontend and backend technologies, this application collects data on various metrics from the student body such as such as age and GPA and is able to securely compute certain calculations over these inputs to gather useful insights. By splitting data into encrypted shares that are stored across multiple databases, the system ensures that no single party can access comprehensive personal data, thereby maintaining privacy and security. This paper goes in depth about the implementation and design of this application and the process followed to ensure secure computation.**

I. Introduction

In today's age, educational institutions around the world have an increased amount of opportunity to gather data. With rapid advancements in technology, data collecting is now more cost effective than it was in the past, which prompts communities to make decisions based on gathered statistics and solid evidence. Embracing this framework allows universities to stay competitive, appeal to the quality-of-life of students, and optimize learning environments. It also allows for personalization, increased efficiency and proper budgeting of effort in regards to issues in a community. This style is common in society and also applicable to any entity that has a community that defines the structure of its organization.

Though, while the shift towards data-based decision making is revolutionary, it's also important to note that if you want to make considerable improvements, data-collection alone needs to be optimized. The most meaningful changes are rooted alongside the most meaningful data, which tends to be sensitive data. Data that, in the wrong hands, could be misused or taken advantage of. This type of data often pertains directly to the challenges presented inside of the community. This can include things like economic disparities, social justice issues, or the conditions of marginalized groups. Understanding this leads to more inclusive policies and interventions that cater towards those affected. The challenge presented to these organizations is: How do we gather this sensitive data and also protect the security and integrity of the members?

Before the big tech and data boom, combining data from multiple sources to get meaningful information was challenging. Methods would require data to be centralized for analysis, which raises a lot of privacy concerns due to data breaches and unauthorized access. Not only that, but sharing data from entity to entity also poses an increased threat of something like that happening. A lot of companies would not risk the legal ramifications that would result from a lack of privacy preserving techniques.

In light of these challenges, it is clear that older methods of data collection and analysis are no longer sufficient when dealing with sensitive information. This is where methods like Multi-Party Computation come into play. MPC allows a way to gather and analyze sensitive data without compromising individual privacy. By distributing the data among multiple parties, this ensures that no single entity has the complete dataset. This also addresses concerns about data breaches and unauthorized access.

II. Related Work

In regards to research on data analytics and privacy preserving techniques, there are studies on how to implement the current techniques of Multi-Party Computation on sensitive data inside of the medical field. There's a paper published in 2018 called "Enabling Analytics on Sensitive Medical Data with Secure Multi-Party Computation". This paper showcases how Multi-Party Computation allows for the study of data without the need to share raw data in the medical field, which brings up instances where they use cryptographic methods to help solve the challenge of combining sensitive data from outside sources. Health laws are extremely strict when it comes to privacy regulations, so being able to comply with privacy laws as well as study sensitive data is critical. Meilof Veeningen, a prominent researcher in the field of cryptosystems at Eindhoven University brings up the concept of rising costs in the realm of healthcare. He

says the solution to this is to access the hidden existing medical data that we already have available. “The use of knowledge that is hidden in existing medical data is seen as the fastest, least costly, and most effective path to improving people’s health and may lead to cost savings of over $250M in the US alone”. In the passage, they bring up methods like data mining on sensitive data or PPDM. Another technique they bring up which is similar to what we’re incorporating in our own project is the ability to share secrets via a large random number “r”. They have multiple parties each calculating said statistic, and adding it alongside the random number r before passing it onto the next party. After subtracting the random numbers from the subtotal, the outcome results in a sum of the subtotal from all the parties.

Meilof Veeningen also brings up one of his pilot projects towards optimizing physical movement and workflow in a hospital setting. By tracking the location data of relevant staff and other workers throughout their shift, viewers can optimize allocation of the workforce. The concern with this is that location data would have to be confined to the bounds of the hospital because sharing location data with management could be considered a breach of privacy. Multi-Party Computation covers that and allows for analytics to be performed without revealing individual data, and any possibility of secondary use of data is eliminated.

III. Solution Overview

That’s where our project comes in. Our project consists of creating an MPC library that will implement MPC to gather sensitive information without revealing individual data. By using Multiparty Computation, we allow multiple parties to compute functions over their private inputs at the same time without revealing the information they placed to others also submitting their own data. Here’s how it works: Each of our data points are split into three shares, which when added together give the original value. This allows us to maintain privacy of each datapoint as the database will not be able to learn anything about the original data from a single share. The shares are then separately stored into different databases to practice decentralized storage, or an additional layer of security, because you don’t want the shares to interact with each other, possibly revealing the actual data. Calculations consist of operations supported by the created MPC library to include generating triples, computing masked values, computing averages, evaluating beaver triple multiplication and combining shares. In case of any specific part of the program being compromised, the information is split apart and incomplete, preventing any entity from gaining access to the complete information. This framework ensures the collection and study of data as well as the security of individuals’ identity and information. As for the current state of this program, it is geared towards an educational institution, with the calculations and frontend being centered around data such as financial aid income, GPA, and other student body facts. But as for future work, we could implement this process with any sort of numerical data. Our MPC framework can be scaled to handle a larger amount of data depending on what it’s used for and potentially evolve to be an application that can shift towards any numerical data inputted. Theoretically, as long as you have an algorithm to mask and split the data, you can place it onto any compatible metric. Also, with how simple our frontend is, it allows the user experience to be simple and less prone to errors. Outside institutions could easily adopt and integrate Multi-Party Computation libraries into their existing data collection and workflows without worrying about data accuracy and compatibility. This allows admins to generate their own visualizations on the trends they see and be able to incorporate the backend into any frontend template they choose to create.

In regards to security, this MPC-based solution adheres to regulatory requirements such as GDPR, CCPA, and FERPA in educational settings. All the data is encrypted whilst being executed inside of the functions and also whilst it’s at rest.

IV. Design

The program is set up so that the backend makes up most of the core components, alongside a frontend that allows for basic interaction and simplicity. It consists of a very simple HTML form that requests user input and parses it for the backend server. Our main backend consists of Google Cloud functions that use a leader-follower architecture for our MPC computation. Each party will have an associated connection to a database that they will be handling, allowing for all three shares of data to be accessed. When developing the MPC backend, the team discussed the benefits and drawbacks of using cloud functions over distinct servers running on top of each database. Cloud functions were easier to set up and more dynamic, allowing for the development of extra features such as encryption of beaver triples and calculation of correlation.

This code defines a class called `MPC_Functions` that provides functionality for secure multi-party computation (MPC). MPC is a cryptographic technique that allows multiple parties to jointly compute a function over

their private inputs without revealing the inputs to each other.

The class includes several methods for generating and manipulating secret shares, which are distributed pieces of data that can be combined to reconstruct the original value. The `generate_shares` method takes a value and the number of parties and generates random shares that sum up to the original value. The `calculate_sum_of_shares` method computes the sum of the shares modulo a large prime number P. The class also includes methods for generating Beaver's triples, which are used in MPC protocols to perform secure multiplication. The `generate_beavers` method generates random shares of two values a and b, as well as shares of their product c. The `generate_beaver_mask` method is used to generate masks for the input shares put in, and the `beaver_compute` method performs the secure multiplication using the Beaver's triples and the masked input shares. Overall, this code provides a foundation for implementing MPC protocols in Python, allowing multiple parties to perform secure computations on their private data without revealing it to each other.

The backend consists of the ability to manipulate the data received from the frontend which is then used to execute computation with the detailed cloud functions. These cloud functions are presented for three parties in the cloud functions directory alongside the program to insert the shares into the database. The cloud functions in the `party1`, `party2`, and `party3` directory are organized in such a way that allows party 1 to be the leader, orchestration cloud functions that are located inside of party 2 and party 3. Some consequences may arise from having party 1 managing the entire process of MPC calculation so there is extra encryption to prevent party 1 from learning anything it should not have access to.

Mean Computation

The most basic operation of our MPC protocol is calculating the mean for any statistics. This is achieved by calling the `calculate_mean` cloud function that is located inside of party 1, which calls the `party2_sum` and `party3_sum` to get the aggregated sum of the party. This value tells us nothing about the individual shares located inside of the databases associated with each party, maintaining the security of the protocol. Next, party 1 will sum these three aggregated shares and divide by the number of datapoints in the database.

The process of calculating the mean using the MPC protocol ensures that the individual shares remain confidential and are not revealed to any single party. By distributing the computation across multiple parties and only sharing the aggregated sums, the protocol maintains the privacy of the underlying data. The calculation begins when the `calculate_mean` function is called, located in party 1's cloud functions. When party 1 receives the aggregated sums from party 2 and party 3 through the `party2_sum` and `party3_sum`, it combines these sums with its own aggregate sum. The resulting value represents the total sum of all the shares across the three parties. However, this aggregated sum alone does not provide any information about the individual shares or the original data points. To obtain the final mean value, party 1 divides the aggregated sum by the total number of data points in the database. This division operation is performed locally by party 1 and does not require any further communication with the other parties. The number of data points is considered public information and does not compromise the security of the protocol. By performing the division step locally, party 1 ensures that the final mean value is computed without revealing any sensitive information to the other parties. The result of this computation is the mean of the data points, calculated securely using the MPC protocol. It's important to note that throughout this process, the individual shares and the original data points remain protected within their respective databases. The MPC protocol allows for the computation of the mean without any party having access to the complete dataset or the individual shares of the other parties. This ensures the privacy and security of the sensitive information while still enabling collaborative computations across multiple parties.

$$\bar{x} = \frac{\sum_{i=1}^{n} s_{1,i} + \sum_{i=1}^{n} s_{2,i} + \sum_{i=1}^{n} s_{3,i}}{n}$$

Standard Deviation Computation

Our next operation, standard deviation, uses the previously mentioned calculation so our first step is to get the mean of the statistic. To calculate the standard deviation we need to find the average of the squared differences of each datapoint from the mean, requiring beaver triple multiplication. So a helper function called `generate_beaver_triples` is used to generate beaver triples equal to the number of multiplication operations needed to be done (a unique beaver triple for each multiplication). As party 1 will be handling the transaction of information, including beaver triples, the share belonging to party 2 and party 3 will be encrypted under their respective secret key. This allows for the protocol to do shared multiplication while ensuring the

privacy of the data.

Code Implementation

```
pk1 = os.environ["PK1"]
pk2 = os.environ["PK2"]
triples = []
for _ in range(triple_count):
    sealed_box_1 = SealedBox(PublicKey(
        public_key=bytes.fromhex(pk1)))
    sealed_box_2 = SealedBox(PublicKey(
        public_key=bytes.fromhex(pk2)))
    beaver_triples = MPC_Functions.
        generate_beavers(3)
    a_shares = [beaver_triples[0][0],
sealed_box_1.encrypt(beaver_triples[0][1].
    to_bytes(length=4, byteorder="big")).hex(),
     sealed_box_2.encrypt(beaver_triples[0][2].
    to_bytes(length=4, byteorder="big")).hex()]
    b_shares = [beaver_triples[1][0],
        sealed_box_1.encrypt(beaver_triples
        [1][1].to_bytes(length=4, byteorder='
        big')).hex(), sealed_box_2.encrypt(
        beaver_triples[1][2].to_bytes(length=4,
         byteorder="big")).hex()]
    c_shares = [beaver_triples[2][0],
        sealed_box_1.encrypt(beaver_triples
        [2][1].to_bytes(length=4, byteorder='
        big')).hex(), sealed_box_2.encrypt(
        beaver_triples[2][2].to_bytes(length=4,
         byteorder="big")).hex()]
    triples.append({"a_shares": a_shares, "
        b_shares": b_shares, "c_shares":
        c_shares})
```

Multiplication Protocol

The next part of multiplication involves computing the shared masked values of the numbers we are multiplying. This is achieved by invoking the `party2_beaver_mask`, `party3_beaver_mask`, computing the masked values for party 1, and subsequently computing the shared value. Afterwards, party 1 computes its aggregated z-share. Party 1 then calls `party2_beaver_compute` and `party3_beaver_compute`, and sums up all the aggregate z-shares. The final step is to return the sum divided by the number of data points and square rooted. All of these functions utilize the `MPC_Functions` as their underlying calculation.

Correlation Computation

Our final operation, correlation, uses both mean and standard deviation as input. The calculation requires multiplication between two statistics, resembling a dot product of values. The remaining calculations are constant multiplications and multiplication between public values. When calculating correlation between x and y, we used a similar protocol used for multiplication as in when calculating standard deviation.

Correlation computation requires:

- The dot product
- The product of the mean of x with the sum of shares of y
- The product of the mean of y with the sum of shares of x
- The product of the means of x and y
- The product of the standard deviations of x and y

Perform this calculation:

$$\frac{X \cdot Y - \mathrm{MX_Y} - \mathrm{MY_X} + \mathrm{MX_{MY}}}{\mathrm{SDX_{SDY}}}$$

Where:

- $MX_Y = (\mathrm{X\ mean})(\mathrm{Y\ sum})$ represents the product of X's mean and the sum of Y.
- $MY_X = (\mathrm{Y\ mean})(\mathrm{X\ sum})$ represents the product of Y's mean and the sum of X.
- $MX_{MY} = (\mathrm{X\ mean})(\mathrm{Y\ mean})$ represents the product of the means of X and Y.
- $SDX_{SDY} = (\mathrm{X\ sd})(\mathrm{Y\ sd})$ represents the product of the standard deviations of X and Y.

This equation describes how to compute the correlation between two variables x and y by considering their means, sums of shares, and standard deviations.

V. Challenges

Developing the frontend had challenges in formatting its styling, leading to difficulty in making the interface friendly and clean. This included the need to make it easily navigable and keep the functionality of the form. Working with HTML to convey the purpose of the project through prompts, which tell where the user should enter their information was crucial to the project. In addition, the need for data validation of the entries was crucial. To do data cleanup and filtering on the backend, adding validation checks to the assignment was needed to ensure that malicious users would not enter exceedingly large values or extremely small values. This had to be communicated with the backend to figure out what the backend needed in terms of data entry. Having compatibility across multiple browsers was crucial to the project too. Ensuring that the webpage was accessible on devices of different formats was important, as our survey users needed accessibility. Handling the interface in order to provide comprehensive errors when the user possibly mistakenly inputted data wrong meant conveying this as error messages. Displaying those error messages in an informative and interpretable way was important.

When developing the cloud function backend, many challenges came up when it came to the implementation and testing of the calculations. Before working on the

cloud functions, the main MPC_Functions library was thoroughly tested with examples of calculating mean, standard deviation, and correlation. These tests were proof of concept that the arithmetic gates were functioning properly and the output data points were close to the expectation, so everything looked fine during that point, but the development of cloud functions and their interactions came with problems such as network-related issues and difficulties in connecting all the resources. Many of our problems stemmed from the complexity of setting up and configuring the network infrastructure to ensure communication between the various components of the system. Also, establishing secure and reliable connections among the cloud functions, databases, and other resources proved to be a little difficult, requiring careful planning and troubleshooting to overcome the network-related hurdles and achieve a fully integrated and functional backend system.

Throughout the development process, we encountered various minor challenges and inconveniences that required additional effort and learning. One general challenge was our limited prior experience in storing data in the cloud and working with servers. As a team, we had to navigate the complexities of cloud infrastructure and familiarize ourselves with the intricacies of Google Cloud Platform.

Even the initial step of configuring Google Cloud to enable project sharing among all five team members proved to be a tedious and time-consuming task. We had to carefully set up the necessary permissions, roles, and access controls to ensure seamless collaboration while maintaining the security and integrity of our project. This process involved a steep learning curve, as we had to understand the various components and services provided by Google Cloud and how they interacted with each other.

VI. Future Work

In addition to simply adding the ability to collect additional metrics from students, there are multiple paths that could be taken into account for future work. One is the ability to calculate linear regression, which is a function that we were never able to fully implement but could be made possible using advanced MPC calculation techniques. Another way to make this system even more secure would be to introduce differential privacy, adding noise to the data set to further mask the input data and reduce the likelihood of being able to obtain any data from the shares. If this program was expanded to a larger scale, access controls could be implemented to restrict data access based on user roles depending on the implementation of whatever company is using the MPC library. We could implement audit trails and logging mechanisms similar to the ones discussed in class to track data access and system activities. This could ensure compliance with privacy laws. Furthermore, this project could be expanded to a wider scale, where not only university students but members of other large institutions and corporations can implement this system to securely store data about individuals and still be able to gather valuable insights. Lastly, this project could also benefit from being able to securely store fixed-point (or float) value data types. At the moment we are multiplying the GPA values by 100 and dividing the aftermath by 100. However, a way to inherently store float values would immensely increase the potential use cases of this project.

Contributions of Team Members

Kevin - Contributed to the web application project by focusing on front-end development and integrating sensitive data handling using Google Cloud. Working in HTML, CSS, and JavaScript, along with integration of backend systems with Firebase.

Adam - Helped create the cloud infrastructure for the backend computations, wrote official documentation for API endpoints that serve cloud functions.

Praj - Implemented MPC calculations in cloud functions, developed `MPC_Functions` library, wrote MPC calculation documentation.

Swagat - Primarily worked on drafting the initial cloud function in Python responsible for inputting data and outputting the slope and intercept of its linear regression model.

Andre - Helped to draft official documentation on the report and work logistics, and managing the integration of simulated data.

Project Repository

Our project code is hosted on GitHub. The main branch contains all frontend and backend code as well as this document. You can access it at the following link: https://github.com/adamdoyle630/comp590-mpc-final.

References

[1] The code written for the MPC functions is modeled after the MPC lab provided in class.

[2] Veeningen M;Chatterjea S;Horváth AZ;Spindler G;Boersma E;van der Spek P;van der Galiën O;Gutteling J;Kraaij W;Veugen T;, Meilof. "Enabling Analytics on Sensitive Medical Data with Secure Multi-Party Computation." Studies in Health Technology and Informatics, U.S. National Library of Medicine, 2018, pubmed.ncbi.nlm.nih.gov/29677926/.