Federated Calculations

Data Privacy and Security - CS528 Fall, 2021

Isaias Rivera

Dec 7, 2020

Contents

Team Description **1**

Application **1**

Dataset **1**

Privacy and Security Techniques **2**

Implementation **3**
- Major Libraries Used . . . 3
- Class Object . . . 3
- User object . . . 4
- Communication . . . 4
- Running a Calculation . . . 4

Testing **5**
- Varying Epsilon . . . 5
- Varying HE Key Size . . . 6

Summary **8**
- Improvements . . . 8

Team Description

The team is only myself. I am a third year undergraduate computer science student doing Co-Terminal at Illinois Tech. I value my online privacy greatly, which is why I chose to use a dataset that involved querying user data from their devices.

Application

The ultimate goal with this project was to have the capability of running calculations over an entire user base. This means the server, or whatever entity is running the calculation, would not know any intermediate values of the computation. The only thing that should be revealed is the final computation to the entity requesting. This allows a requester to run a calculation over an entire user base without compromising any user's privacy.

This project also assumes that both parties follow protocol, as either a malicious user or requester could either report false data ruining a calculation or preemptively decrypt a running calculation.

Dataset

Fitbit Fitness obtained from Kaggle includes personal fitness tracker data from 33 Fitbit users. The data includes things like daily calories burnt to hourly steps taken. Each user is separated with their own data, where no other object/entity is allowed directly access this data except for testing purposes.

Dataset Link: kaggle.com/mjazzy/fitbit-fitness-bellabeat-high-tech-company

Privacy and Security Techniques

When a attribute is to be used for a calculation, Laplacian noise is added to the value. Noise is added to each value before a calculation is done as post processing of values does not affect the privacy budget. Because calculations can use multiple attributes, it is also ensured that privacy parameter is split up between each attribute.

Additionally, to ensure the entity requesting the calculation does not see any intermediary values, the calculations are performed on HE values as it is passed from user to user, as shown in figure 1.

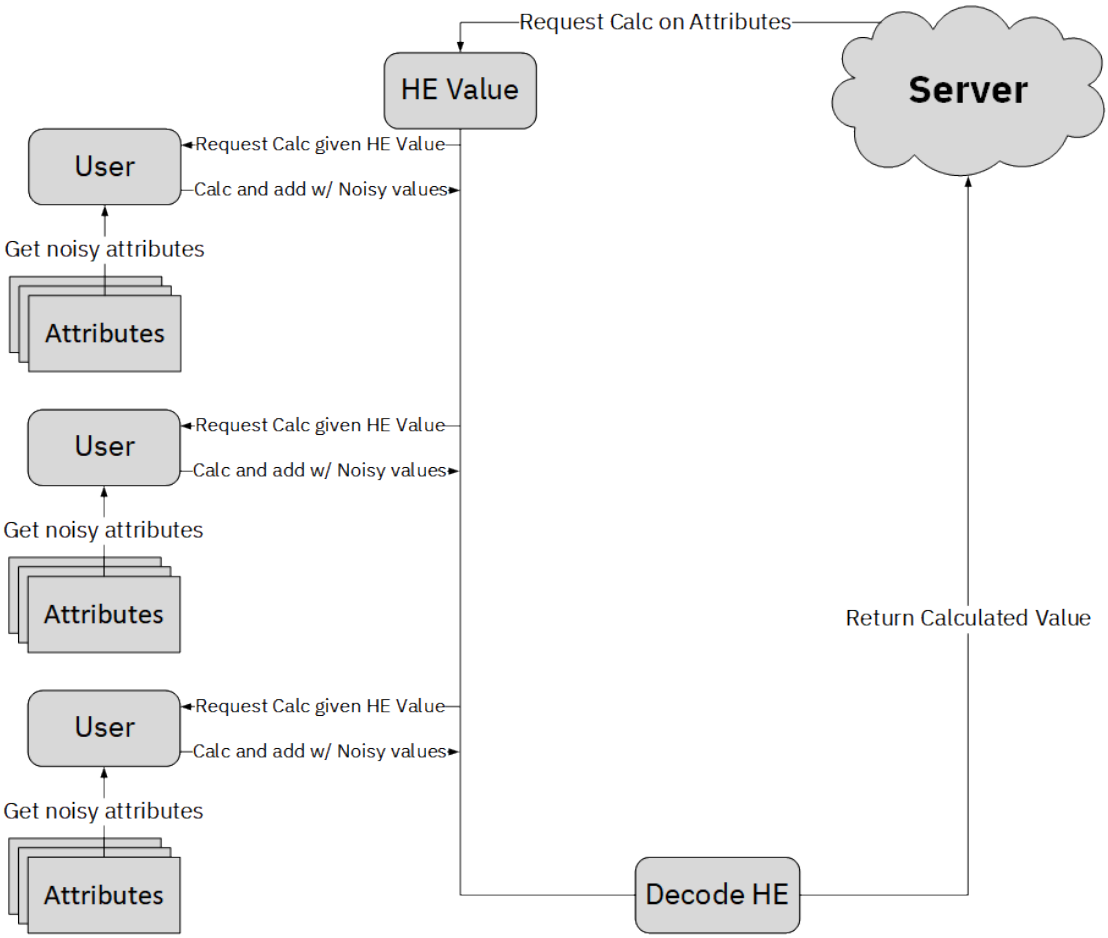

Figure 1: The lifecycle of a server request to a user base

Implementation

This project was made using Python 3.10

Major Libraries Used

- phe 1.4.0
  - Used for Paillier homomorphic encryption
- diffprivlib 0.5.0
  - Used for the laplacian noise mechanism
- pandas 1.3.4
  - Used for general management of data
- pathos 0.2.8
  - Used for multiprocessing when generating the user base

Class Object

The class object is only used to simulate someone requesting a value from the user base. It also takes care of the homomorphic encryption and decryption of the calculated value.

Listing 1 An example of a single User in the dataset

```
--------[ 1624580081 ]--------
  daily
    TotalSteps                 31
    TotalDistance              31
    TrackerDistance            31
    LoggedActivitiesDistance   31
    VeryActiveDistance         31
    ModeratelyActiveDistance   31
    LightActiveDistance        31
    SedentaryActiveDistance    31
    VeryActiveMinutes          31
    FairlyActiveMinutes        31
    LightlyActiveMinutes       31
    SedentaryMinutes           31
    Calories                   31
  hourly
    Calories            736
    TotalIntensity      736
    AverageIntensity    736
    StepTotal           736
```

User object

The dataset is comprised of various fields grouped up into different intervals and types. This project included the daily, hourly, sleep, and heart entries from the dataset, where each entry has it's own fields. These are values that would hypothetically be on a Fitbit device. Listing 1 shows an example of a user after it has been filtered out of the dataset where each number represents the number of entries for that specific field.

Each user has the ability to output a list of fields given the privacy parameter ϵ, where it will appropriately add laplacian noise to each field in the outgoing list. This is done with the `UserRequest` class.

Communication

Currently, communication between Server and User is done on the same instance, however, both objects do not access attributes directly from each other.

Listing 2 Example calculation

```
users: list[User] = dataset.get_dataset(True)  # Get all users as objects
server = Server(users, epsilon=10, key_size=1024)  # Initialize a server with privacy p

# Example calculation given a list of attributes, determined upon request
def test(pub: PubKey, acc: EncryptedNumber, usr: list[Number]) -> EncryptedNumber | bool
    calories = usr[0]
    mod_active_dist = usr[1]

    if mod_active_dist != 0:  # Check for an actual value
        acc = acc + EncodedNumber.encode(pub, calories * mod_active_dist)
        return acc

    return False

# Run function with the fields "Daily calories" and "Daily moderately active distance"
request_ldp = server.requestAction([FRAME.DAILY.CALORIES, FRAME.DAILY.MODERATELY_ACTIVE_
```

Running a Calculation

Listing 2 shows an example of running a custom calculation on the entire user base. This calculation multiplies and aggregates the 'Daily calories' and 'Daily moderately active distance' fields of each user and then gets the average of that aggregated value.

Testing

The following tests show how varying the privacy parameter ϵ and the key size for homomorphic encryption affects the accuracy and time to calculate using the same test function as in Listing 2

An additional function is used to time the test function as follows

```
def timedTest(eps: float, key: int):
    t0 = time.time()
    test_run(eps, key)
    t1 = time.time()
    return (key, t1 - t0)
```

Varying Epsilon

Epsilon was changed from the range 0.1 to 10 with a step of 0.05. The key size was kept at 512.

The final value output is the error between the actual value and the value modified for ldp. Figure 2 plots these values.

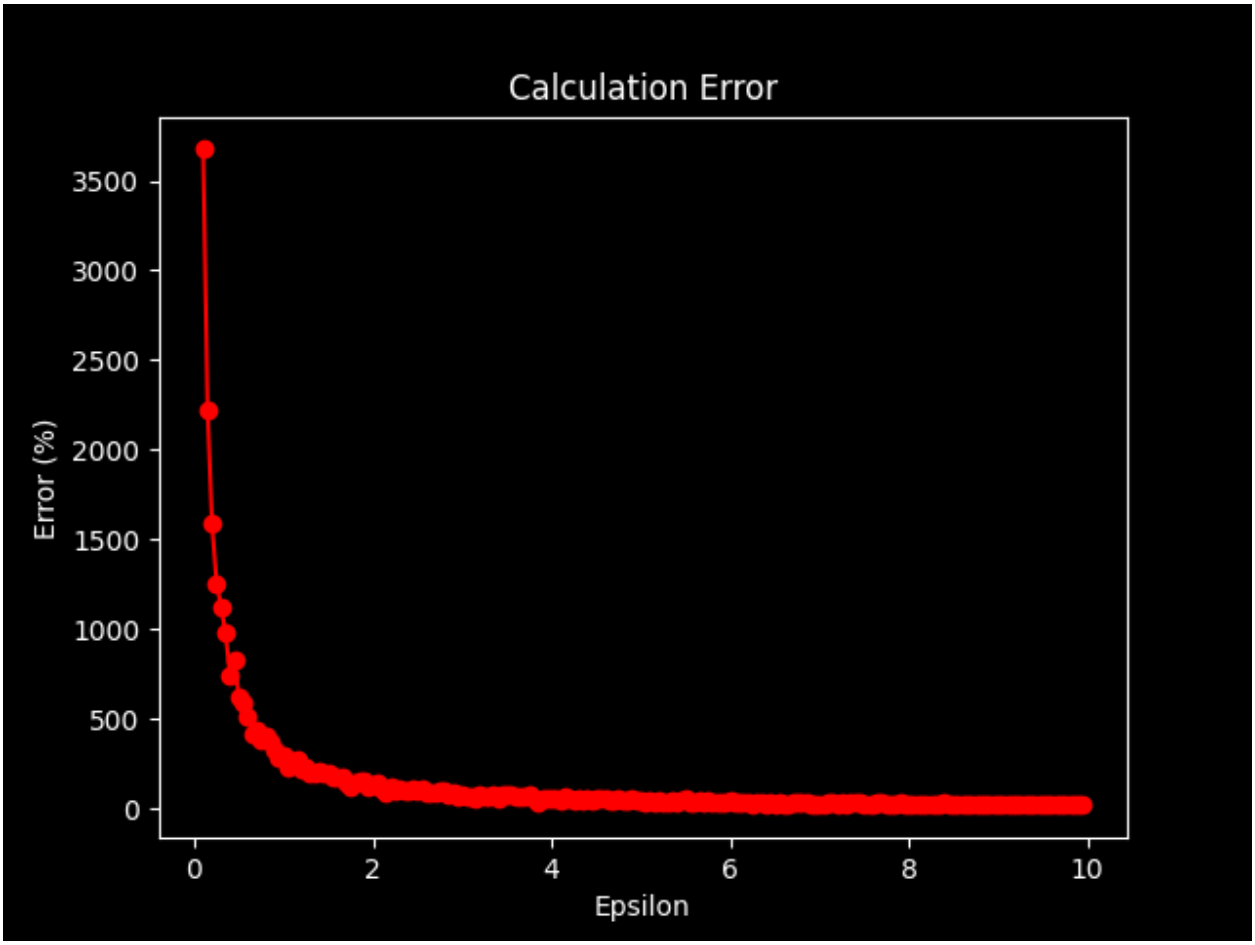

Figure 2: The lifecycle of a server request to a user base

Varying HE Key Size

The key size for HE was changed from the range 256 to 4096 with a step of 64. Epsilon was kept at 5.

The final value output is the time it took to calculate. Figure 2 plots these values.

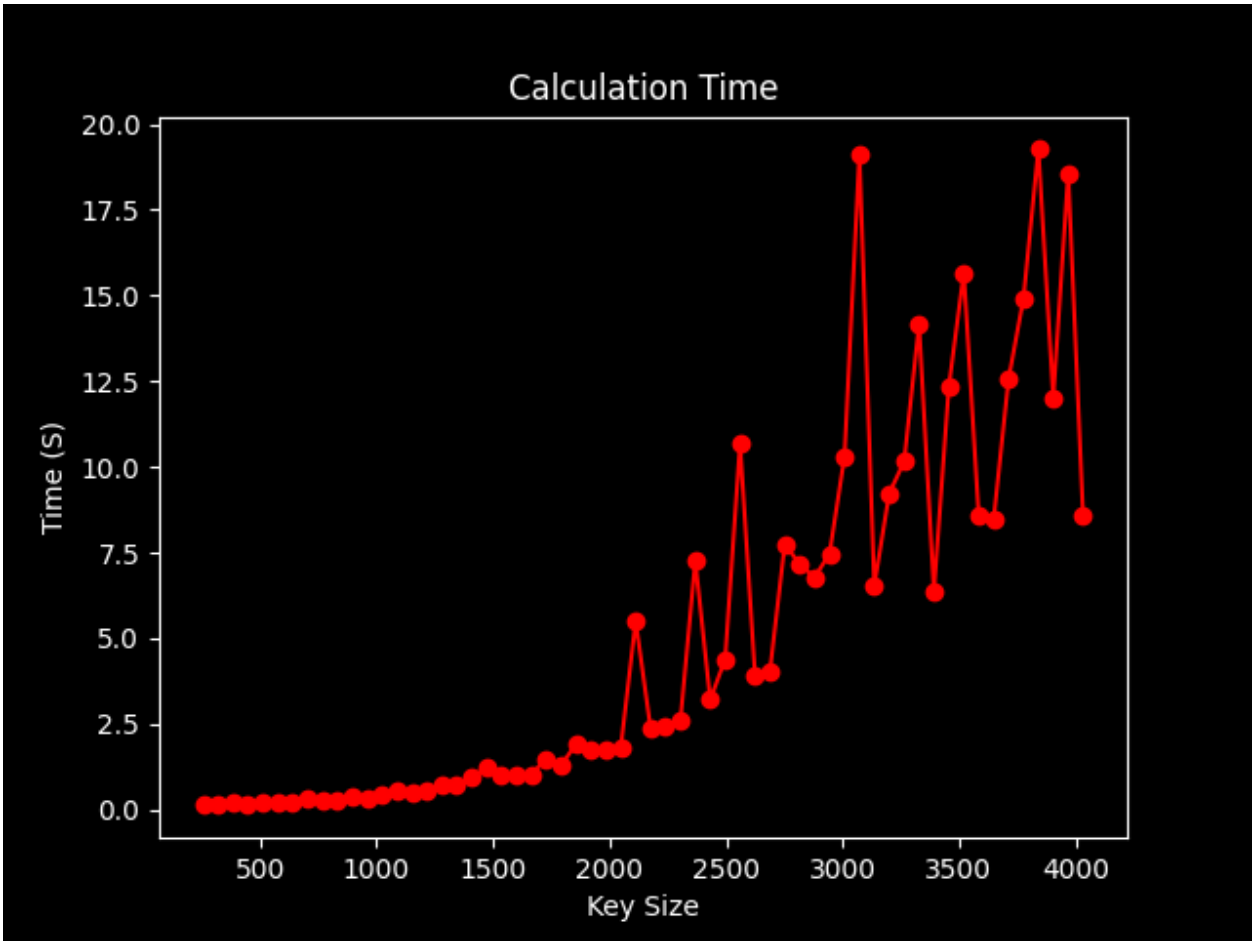

Figure 3: The lifecycle of a server request to a user base

Summary

Overall, I believe I have demonstrated how calculations can be run over an entire user base while still respecting standard practice when it comes to maintaining user privacy. Additionally, the inclusion of homomorphic encryption does not add a significant amount of overhead when the key size is reasonable. However, I imagine this overhead would be multiplied if it were to be run on a low power device and if it were to actually be transmitted between devices.

Improvements

In addition to a server being able to run calculations on a user base, I also would want each user to be able to compare with each other without revealing each other's values. This might be accomplished with garbled circuits, however that might be too inefficient. Additionally, this could be expanded to federated learning, where each user could create a local model and be grouped with a server model.

Furthermore, currently, the encrypted value from the current running request is passed from server to user to server and then another user. It would be best if the requester just passed said encrypted value to a user and then have that user pass it to another user until all users had aggregated their calculation. This would further prevent a requester from looking at intermediate values.