Mandatory exercise set 2

Adam Rose - adjr@itu.dk - 30-10-2023

Problem Description

Alice, Bob and Charlie are participating in a medical experiment in their local Hospital where they need to provide data for training a Machine Learning (ML) model, which will later be used to diagnose patients. In this setting, the Hospital runs a central server that collects data from all patients. The Hospital is trusted to execute a data collection protocol established by the Hospital and pre-agreed among all patients taking part in the experiment, without actively trying to cheat on patients and extract and/or reveal their data. However, since the experiment deals with specially sensitive personal data (i.e. health data), neither the patient nor the hospital want plaintext patient data to be directly processed and/or stored by the Hospital. Moreover, the patients do not know each other, and consequently do not trust each other to process and/or store their data. Nevertheless, the patients do trust each other to follow the protocol established by the Hospital. As usual, the patients and the Hospital communicate over insecure networks (i.e. the Internet), where eavesdroppers may try to obtain private data and/or compromise the accuracy of the experiment by tampering with data while it is transmitted.

The restrictions in the scenario above leave the Hospital and patients in a tight spot, since standard ML algorithms require access to plaintext data in order to train a model. Luckily, the researchers in the hospital are collaborating with a team of data scientists and security experts who are all aware of the latest advances in Federated Learning, which allows for distributed training of models using data held locally by different users. These experts have designed a Federated Learning algorithm that supports a technique called Secure Aggregation, which allows the algorithm to train a model from aggregate data encoding all the plaintext data individually held by each patient into a single value. In particular, in this scenario, it is sufficient for the patient data to be aggregated by summing all individual patient's values into one final aggregate value, which needs to be made available to the Hospital. Using this technique, neither the patients nor the Hospital get access to each patient's individual plaintext data.

Design and implement a solution that allows for the 3 patients interacting among each other and with the Hospital in the scenario described above to compute an aggregate value of their private input values in such a way that only the Hospital learns this aggregate value. Consider that all individual values held by patients are integers in a range [0,. . . ,R] and that the aggregated value is the sum of all individual values, which is also assumed to be in the same range. You must describe an adversarial model (or threat model) capturing the attacks by an adversary who behaves according to the scenario described above, explain how your solution works and why it guarantees security against such an adversary.

Hint: Secure Aggregation for Federated Learning is a real-world practical technique.

Deliverables:

- A written report describing the adversarial model you are working on, describing the building blocks of your proposed solution, how they are combined in your final solution and why they guarantee security against the adversary you describe.
- An implementation of your solution in a programming language of your choosing, along with clear instructions on how to compile and run it (potentially added to the report or to a separate Readme file).

Submission Instructions:

- Submit only the PDF file with your report and the file containing your code. Do not submit whole folders containing metadata, auxiliary IDE files or anything else than the code and report.
- Please name your submission clearly using your Name/Student ID, e.g. Jane Doe - 36476832.zip, Jane Doe - 36476832 - Report.PDF, Jane Doe - 36476832 - code.c, Jone Doe - 36476832 - Readme.txt. This makes grading faster, so that you get feedback on your hand-in faster.

Solution

Adversarial Model

The parties taking part in the protocol do not trust each other with their personal data, thus the protocol should be secure with a *dishonest majority*. Furthermore, the protocol should also be secure under the assumption that the adversary is *adaptive*. It is however assumed that the adversaey is *passive*, and doesn't stray away from following the protocol.

Finally, the adversary has full control of the network, and is thus a Dolev-Yao adversary. It may manipulate the network in any way it wants but can't break encryption.

Based on this adversarial model, here is a summary of the security requirements and assumptions for the protocol:

- None of those who take part of the protocol stray away from it
- It should not reveal the data of one patient to another
- It should not reveal the data of any single patient to the hospital
- The protocol should ensure the confidentiality of messages sent between patients/hospital
- The protocol should ensure the integrity of messages sent between patients/hospital
- The protocol should ensure authenticity of messages sent between patients/hospital

Protocol Design

To fulfil the requirement of not revealing data to any other patient/hospital, it is necessary to use *Secure Multi-Party Computation*, which allows one to distribute a computation across several machines without revealing any of the individual pieces of data. In this case, the computation is *Secure Aggregation*, which simply involves summing all the patients data together. This can be done using *n-out-of-n additive secret sharing*, which divides a patients data into n shares. The data can only be recovered once all n shares are added together. This means that the patient won't share their secret data unless they give away all the shares.

To fulfil the requirements of confidentiality, integrity and authenticity, *Transport Layer Security* is used. It addresses each one of these concerns, and is tried and tested.

- It addresses confidentiality through symmetric cryptography.
- It addresses integrity through MAC.
- It addresses authenticity through digital certificates.

The steps that the hospital takes are the following:

1. Wait for all the patients to register themselves. I make the assumption that if a patient registers with the hospital, they are taking part in the protocol.
2. Each time a patient registers, send their port to all the other patients.
3. After waiting a certain amount of time, tell all patients that the computation can now start.
4. Wait till it receives shares from all the patients.
5. Add the shares together: $s_0 + ... + s_n$, where n is the amount of patients.

The steps that each patient takes are the following:

1. Choose a number d that represents the patient data in the range $[0, ..., R/n]$, where n is the amount of patients. It has to be within this range since the result of aggregating all the data must not exceed R.
2. Register their port at the hospital.
3. Keep track of ports of other patients, sent by the hospital.
4. When signal from hospital to start computation received:
    1. Divide their secret into n parts, where n is the amount of registered patients (including themselves).
        - This is done by choosing $n - 1$ random positive integers within the range $[0, ..., N]$.
        - The final share is calculated with $s_n = d - (s_0 + ... + s_{n-1})$, where $s_0 ... s_{n-1}$ are the previously chosen shares, and d is the patient data

2. Send $n-1$ shares to other patients.
3. Wait till shares from all other patients have been received.
4. Add received shares and own unsent share together.
5. Send the result to the hospital.

Of course, all communication happens over *TLS*.

This protocol works under a dishonest majority, but only in cases where at least 2 participants (including the hospital) are honest.

Implementation

I implemented my solution using Go, which has the `http`-package. This package has built in functionality for serving with *TLS*. I created HTTP-servers for both the patient and hospital. They communicate with each other through HTTP-requests.

A few shortcuts i took for the implementation of the protocol:

- I do not handle cases where someone strays away from the protocol
- I use the same digital certificate for all patients, and the hospital. In reality each would have their own certificate.

The hospital client and server is implemented in `hospital/hospital.go`.
It has the following endpoints:

- `/patients` - where it accepts a POST request with the port of a patient
- `/shares` - where it accepts a POST request with an aggregate share

The patient client and server is implemented in `patient/patient.go`
It has the following endpoints:

- `/patients` - where it accepts a POST request with the list of ports of other patients
- `/shares` - where it accepts a POST request with the share of another patient

How to run the code To run the code it is first necessary to generate a private key and a digital certificate. This is done with the following commands:

- `openssl genrsa -out server.key 2048`
- `openssl req -new -x509 -sha256 -key server.key -out server.crt -days 3650 -addext "subjectAltName = DNS:localhost"`

The files `server.crt` and `server.key` should now be in the root of the handin-folder.

Then, simply run `./main.sh`. This shell script starts 3 instances of patients, and one instance of the hospital.

Example Execution

```
$ ./main.sh
2023/10/30 21:13:36 8080 : Creating hospital server
2023/10/30 21:13:37 8083 : New patient with data = 22
2023/10/30 21:13:37 8083 : Patient registering with hospital
2023/10/30 21:13:37 8083 : Creating patient server
2023/10/30 21:13:37 8080 : Hospital received /patients
2023/10/30 21:13:37 8080 : Hospital received POST /patients
2023/10/30 21:13:37 8080 : Registered new patient at port 8083
2023/10/30 21:13:37 8083 : Registered with hospital, received response code 200 OK
2023/10/30 21:13:37 8082 : New patient with data = 137
2023/10/30 21:13:38 8082 : Patient registering with hospital
2023/10/30 21:13:37 8081 : New patient with data = 158
2023/10/30 21:13:38 8082 : Creating patient server
2023/10/30 21:13:38 8081 : Patient registering with hospital
2023/10/30 21:13:38 8081 : Creating patient server
2023/10/30 21:13:38 8080 : Hospital received /patients
2023/10/30 21:13:38 8080 : Hospital received POST /patients
2023/10/30 21:13:38 8080 : Hospital received /patients
2023/10/30 21:13:38 8080 : Hospital received POST /patients
2023/10/30 21:13:38 8080 : Registered new patient at port 8082
2023/10/30 21:13:38 8080 : Registered new patient at port 8081
2023/10/30 21:13:38 8080 : Sending ports to patients
2023/10/30 21:13:38 8080 : Sending ports [8081 8082]  to 8083
2023/10/30 21:13:38 8082 : Registered with hospital, received response code 200 OK
2023/10/30 21:13:38 8083 : Patient received POST /patients
2023/10/30 21:13:38 8083 : Sending shares to other patients
2023/10/30 21:13:38 8081 : Patient received POST /shares
2023/10/30 21:13:38 8083 : Sent share to,  8081 . Received response code: 200
2023/10/30 21:13:38 8082 : Patient received POST /shares
2023/10/30 21:13:38 8083 : Sent share to,  8082 . Received response code: 200
2023/10/30 21:13:38 8080 : Sent ports to  8083 . Received response code 200 OK
2023/10/30 21:13:38 8080 : Sending ports [8083 8081]  to 8082
2023/10/30 21:13:38 8082 : Patient received POST /patients
2023/10/30 21:13:38 8082 : Sending shares to other patients
2023/10/30 21:13:38 8083 : Patient received POST /shares
2023/10/30 21:13:38 8082 : Sent share to,  8083 . Received response code: 200
2023/10/30 21:13:38 8081 : Patient received POST /shares
2023/10/30 21:13:38 8082 : Sent share to,  8081 . Received response code: 200
2023/10/30 21:13:38 8080 : Sent ports to  8082 . Received response code 200 OK
2023/10/30 21:13:38 8080 : Sending ports [8083 8082]  to 8081
2023/10/30 21:13:38 8081 : Patient received POST /patients
2023/10/30 21:13:38 8081 : Sending shares to other patients
2023/10/30 21:13:38 8083 : Patient received POST /shares
2023/10/30 21:13:38 8083 : Computing aggregate share
```

```
2023/10/30 21:13:38 8083 : aggregate share is  -23
2023/10/30 21:13:38 8083 : Sending aggregate share -23 to hospital
2023/10/30 21:13:38 8080 : Hospital received POST /shares
2023/10/30 21:13:38 8080 : Hospital received share -23 , total of 1
2023/10/30 21:13:38 8082 : Sent aggregate share to hospital, received response code 200
2023/10/30 21:13:38 8081 : Sent share to,  8082 . Received response code: 200
2023/10/30 21:13:38 8081 : Computing aggregate share
2023/10/30 21:13:38 8081 : aggregate share is  305
2023/10/30 21:13:38 8081 : Sending aggregate share 305 to hospital
2023/10/30 21:13:38 8080 : Hospital received POST /shares
2023/10/30 21:13:38 8080 : Hospital received share 305 , total of 3
2023/10/30 21:13:38 Computation finished: The final value is 317
2023/10/30 21:13:38 8081 : Sent aggregate share to hospital, received response code 200
2023/10/30 21:13:38 8080 : Sent ports to  8081 . Received response code 200 OK
2023/10/30 21:13:38 8081 : Registered with hospital, received response code 200 OK
```

$$x + x$$