

Report Project Hand-in 2

Lamberto Ragnolini

Fall Semester, October 22, 2024

1 QUESTION 1!

1.1 QUESTION

Reflect on this scenario in the context of the GDPR: What are the potential issues in having the hospital store plaintext private data provided by patients even if they have consented to participate on the experiment and have their data processed? Would these issues be solved by removing the patients' names from their data before storing it? What are the remaining risks in using Federated Learning with Secure Aggregation as suggested?

1.2 ANSWER

The hospital by storing plaintext data is making a big mistake. Immediately from the fact of speaking about the ability of seeing plaintext data is bad, there are many reasons why the hospital should not do that.

Taken from the slides : "Data processors must obtain consent from data subjects before processing their data and inform of how data will be used." In this scenario the Hospital, as explained in the assignment, has full consent from patientes to process their data, however this DOES NOT give permission to the hospital to avoid giving full protection of to the processed data. By storing plaintext data many problems could arise. Given that, taken from slides : "Data subjects have special personal data: race/ethnicity, political opinion, religious beliefs, sexual life/orientation, health/genetic data." this leads to highly sensitive data managing which must include a massive protection environment. By storing plaintext data imagine a scenario where there is a leak in the database and someone is able to access the sensitive information... They would have direct access to medical records which is really not what we want. If the data would be encrypted then even if the storage would have problems, leakages or gets hacked, there would be another protection layer (encrypted data) that would prevent the attacker from accessing the highly sensitive data, giving that as GDPR states, medical records are classified as highly sensitive data.

All of this could happen and explicit consent from the patientes does not remove the risk of what just explained.

Removing the names from the patientes records would not help, there are many other informations that could reconstruct the patient identity and given that we are also talking about machine learning, we know that machine learning approaches are known for recognising patterns and from multiple information reconsntruct a general one... This would simply slow down the process given that a possible attacker would have to do all of this to trace back the identities.

Also taken from the slides : "Data subjects also have rights to: erasure, access, rectification, restrict processing, portability, object to processing", This would slow down the process of finding records

in the database and adding complexity to the searching itself. Remember WE ARE STILL LEAKING PLAINTEXT INFORMATION, as repeated many times during lectures we want to be able not to show ANY information regarding what we are trying to keep secret.

There are still many risks in using a federate Learning with secure Aggregation, the easiest one to think about is that there could be a slight chance of learning something about the initial input through the aggregation even though it would be very hard it is still possible. The biggest problems I could think of for this approach would be as also discussed in class the lack of assurance that all parties are not corrupted... If one or more parties are corrupted, many bad things could happen i.e sharing of an aggregation with different shares rather than the real ones. Many parties could ally and try to hi-jack the learning and try to learn by themselves the informations or integrity of the data could be compromised by just one party leading to a result that is not the right one...

As explained many things could go wrong, the most probable one is the scenario where one or more parties are corrupted.

Not to leave out the fact that the exchange of information is done through the internet, this could lead also to a possible threat as a Dolev-Yao attacker who can intercept and change the messages and do many other bad things to our shared data, this is to prevent and TLS will be our best friend when dealing with it.

2 QUESTION 2!

2.1 QUESTION

Design and implement a solution that allows for the 3 patients and the Hospital in the scenario above to compute an aggregate value of the patients' private input values in such a way that the Hospital only learns this aggregate value and no patient learns anything besides their own private inputs. Your protocol must also ensure confidentiality and integrity of the data against external attackers. Consider that all individual values held by patients are integers in a range $[0, \dots, R]$ 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.

2.2 ANSWER

2.3 HOW TO RUN THE CODE

There will be 4 files with python code, that are the following :

- /HOSPITAL/hospital_server.py
- /CLIENTS/client_Alice.py
- /CLIENTS/client_Bob.py
- /CLIENTS/client_Charlie.py

The following steps are the steps to follow to be able to run the whole assignment :

- open 4 terminal windows
- first run the hospital server in one of the terminals with the following command -> `python3 hospital_server.py`

- in the other three windows run on each the same command but with the clients server i.e
1st. tw -> python3 client_Alice.py
2nd tw -> python3 client_Bob.py
3rd tw -> python3 client_Charlie.py
AFTER RUNNING ALICE SERVER YOU WILL HAVE ONLY 15 SECONDS BEFORE ALICE WILL START SENDING DATA TO THE OTHERS SO MAKE SURE TO RUN BOB AND CHARLIE WITHIN 15 SEC FROM WHEN ALICE WAS RAN.
- Now just WAIT for the processes to talk to each other and compute all the needed work
- every server will print what is it doing and to who is it sending the data
- THE PROCESSES have timeout functions, meaning that everything runs after a certain time (not that much time in max a min everything will be done)
- THE USER DOES NOT NEED TO DO ANITHING, i.e input any number, everything works by itself
- THE EXPLANATION WILL FOLLOW LATER IN THE DOCUMENT OF HOW EVERYTHING WORKS
- The hospital process will be killed by itself at the end while the clients one needs to be stopped (ctrl + c in tw)

2.4 EXPLANATION

The three client processes have the same exact code, obviously adjusted due to variable names and data to send, but 95% of the code is the same. The hospital code is basically the same with less auxilliary functions given the fact that it does not need to send any data to the clients apart from its public key.

- The most important variable in the code is the following :

```

1      # MOST IMPORTANT VARIABLE, THE LOCAL INITIAL VALUE OF THE PATIENT
2      # integers value in a range [0,...,R]
3      patient_value = 725

```

it is the private value of each client, it is found at the beginning of each client script.

- The parties communicate with each other through socket connections. Given the fact that I took a networking course in my home university I already knew how to handle them but still made some research and some of my sources are coming from : [link](#). Given that is not the focus of the assignment I will move on to explain the other things
- MAIN POINTS -> when the clients server start, they send their public key freely (meaning without encryption but only through TLS) to the other clients and to the hospital. ONLY WHEN the hospital has all the public keys, it will send his to the three patients so that they can start talking to each other and start sending encrypted data that will then be sent to the Hospital itself.

The assignment asks to provide confidentiality and integrity in the whole required process. My approach works in this way, The hospital waits for all the public keys to be exchanged so that later through asymmetric key encryption we can have a channel that communicates through encrypted data other than TLS.

The key exchange is done without any encryption to the public key because how will the other parties know how to decrypt it if they don't have anything to do it yet...??. This is though not a problem because in a possible scenario where a Dolev-Yao attacker is trying to eavesdrop our messages, he would not learn anything from them giving that the public key, as the name suggests, is public, we have no problem in showing that, so the classic MAN-IN-THE-MIDDLE attack would be useless because it will just give the attacker access to our public key.

The following is the code used by the three patients to exchange their public key (code taken from one of the three patient) :

```
1      # This function is the function to exchange Charlie's public key so that the
      # other parties can
2  # communicate with Charlie using asymmetric encryption on the subsequent
      # connections
3  def send_pk(pk, who) :
4
5      # Create a secure SSL context
6      # This loads the certificates which will be asked from the other parties to
      # prove that Charlie is safe
7      context = ssl.create_default_context()
8      context.load_cert_chain(certfile=" ../CERTS/Charlie.crt", keyfile=" ../CERTS/
      Charlie.key")
9      context.load_verify_locations(" ../CERTS/myCA.pem")
10     context.verify_mode = ssl.CERT_REQUIRED
11
12     # connects to who needs to send the pk
13     secure_sock = context.wrap_socket(socket.socket(socket.AF_INET),
      server_hostname='localhost')
14     secure_sock.connect(('localhost', who))
15
16     try:
17         # Serialize the public key to PEM format
18         pem = pk.public_bytes(
19             encoding=serialization.Encoding.PEM,
20             format=serialization.PublicFormat.SubjectPublicKeyInfo
21         )
22         # send the pk
23         secure_sock.sendall(pem)
24         print(f"My public key has just been sent to : {who}")
25     finally:
26         # close connection
27         secure_sock.close()
```

Is what I've said really true though? NO... as seen in class and written on the slides a man-in-the-middle attack can do more than just eavesdrop our messages, it can send additional messages on the channel and can impersonate us by dropping our just sent public key and creating his own making the other thinking that they are talking to someone while instead they are sending data to the attacker...

This destroys what we are trying to ensure, integrity and confidentiality, that's why to prevent this we are using TLS. TLS creates a secure channel communication between the parties, providing confidentiality and integrity. How do we ensure this? With certificates...

```
1      context = ssl.create_default_context()
2      context.load_cert_chain(certfile=" ../CERTS/Charlie.crt", keyfile=" ../CERTS/
      Charlie.key")
```

```

3 context.load_verify_locations("../CERTS/myCA.pem")
4 context.verify_mode = ssl.CERT_REQUIRED

```

The above code is the code that ensures the creation of a safe context on every exchange. Every time a party is trying to exchange data, encrypted or not, TLS provides a safe transport protocol by ensuring that the other party is a certified party, meaning provides a proof of validity through a signed certificate. This is done on EVERY EXCHANGE.

This ensures that no attacker will be able to impersonate any other given that it will not have our certificate as proof to provide to the receiving party.

Once the public key exchange is done, and the Hospital has sent its public key, now every party is able to start with the aggregation algorithm. Every party will split its private value, encrypt it with the previously received public key and send it to the others. Once sent the parties will receive their part, decrypt it with their private key and compute the aggregation locally.

```

0     # This function is the one that sends the actual data meaning the
1     # aggregation value, both the splitted ones and
2     # the locally computed aggregation
3     def send_data(value, who, pk):
4
5         print(f"the value I am sending before encryption is : {value}")
6
7         # prepare data to be encrypted
8         data = str(value).encode('utf-8')
9         # when we receive the key and we store it into the global pk's it will be a
10        # string so we need to
11        # ensure that the public key is in bytes and if not we convert it to bytes
12        if isinstance(pk, str):
13            pk = pk.encode('utf-8')
14
15        # we need the key to be in pem format to encrypt the data
16        pk = serialization.load_pem_public_key(pk)
17
18        # Encryption, as said in class USE LIBRARIESSS
19        encrypted_data = pk.encrypt(
20            data,
21            padding.OAEP(
22                mgf=padding.MGF1(algorithm=hashes.SHA256()),
23                algorithm=hashes.SHA256(),
24                label=None
25            )
26        )
27
28        # Create a secure SSL context
29        context = ssl.create_default_context()
30        # present certificates that will be requested to prove safe identity
31        context.load_cert_chain(certfile="../CERTS/Charlie.crt", keyfile="../CERTS/
32        Charlie.key")
33        context.load_verify_locations("../CERTS/myCA.pem")
34
35        # connects to who needs to send the encrypted data
36        secure_sock = context.wrap_socket(socket.socket(socket.AF_INET),
37            server_hostname='localhost')
38        secure_sock.connect(('localhost', who))

```

```

36     try:
37         # send encrypted data
38         secure_sock.sendall(encrypted_data)
39         print(f"the encrypted value was sent to : {who}")
40     finally:
41         # close connection after every exchange
42         secure_sock.close()

```

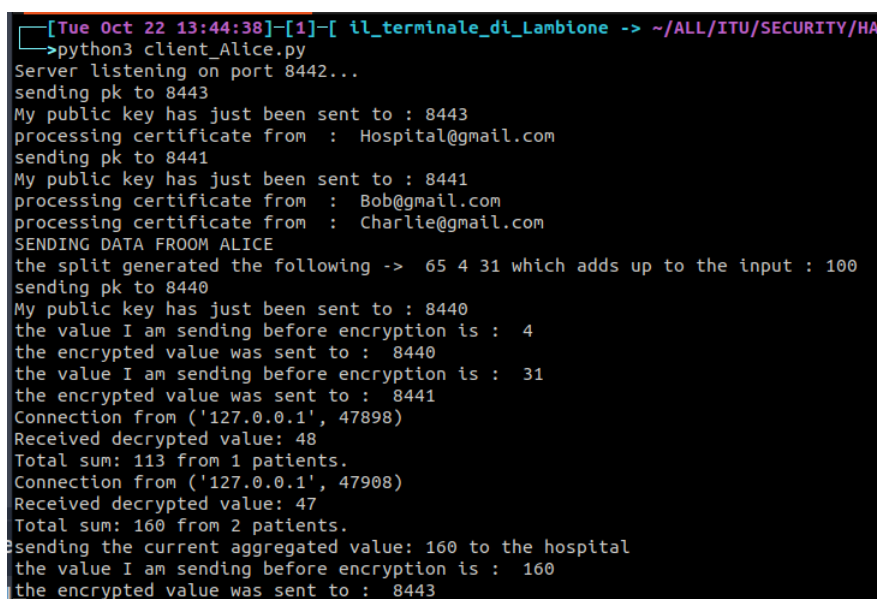
The above code reports the auxilliary function to send the splitted values or the aggregated value to the others. We can see that it encrypts the data with the previously provided public keys, creates a new secure communication channel through TLS and only then is transfers the data. This ensures authenticity, confidentiality and integrity. Even though a man in the middle eavesdrop the message, it will find an encrypted message so he will not be able to understand anything about the content and more than that he will not be able to do anything as explained before such as message injection or anything else because they will be rejected due to the lack of valid certificates by the attacker. This can't happen though given to the TLS nature that prevents exactly this, a possible attacker will not be able to eavesdrop our messages.

The parties will then compute locally their aggregations, sending them always through encryption and secure channel to the Hospital who will be able to compute the single aggregated value it needs.

2.5 CLARIFICATION

What is explained above is the full process of the whole algorithm works and how I handled the different parts, the code is longer but the most important parts have been reported above here with the theory explanation. **THE CODE THOUGH IS WRITTEN WITH DETAILED COMMENTS ON EACH SECTION TO ALLOW A BETTER UNDERSTANDING ALSO OF THE OVERALL IMPLEMENTATION.**

3 RUNNING THE ALGORITHM



```

[Tue Oct 22 13:44:38]-[1]-[ il_terminale_di_Lambione -> ~/ALL/ITU/SECURITY/HAI
>python3 client_Alice.py
Server listening on port 8442...
sending pk to 8443
My public key has just been sent to : 8443
processing certificate from : Hospital@gmail.com
sending pk to 8441
My public key has just been sent to : 8441
processing certificate from : Bob@gmail.com
processing certificate from : Charlie@gmail.com
SENDING DATA FROM ALICE
the split generated the following -> 65 4 31 which adds up to the input : 100
sending pk to 8440
My public key has just been sent to : 8440
the value I am sending before encryption is : 4
the encrypted value was sent to : 8440
the value I am sending before encryption is : 31
the encrypted value was sent to : 8441
Connection from ('127.0.0.1', 47898)
Received decrypted value: 48
Total sum: 113 from 1 patients.
Connection from ('127.0.0.1', 47908)
Received decrypted value: 47
Total sum: 160 from 2 patients.
sending the current aggregated value: 160 to the hospital
the value I am sending before encryption is : 160
the encrypted value was sent to : 8443

```

Figure 1: Log of Alice

```

[Tue Oct 22 13:44:59]-[1]-[ il_terminale_di_Lambione -> ~/ALL/ITU/SECURITY/HAI
python3 client_Bob.py
Server listening on port 8441...
sending pk to 8443
My public key has just been sent to : 8443
processing certificate from : Hospital@gmail.com
processing certificate from : Alice@gmail.com
sending pk to 8442
My public key has just been sent to : 8442
sending pk to 8440
My public key has just been sent to : 8440
processing certificate from : Charlie@gmail.com
SENDING DATA FROM BOB
the split generated the following -> 41 11 48 which adds up to the input : 100
Connection from ('127.0.0.1', 55606)
Received decrypted value: 31
Total sum: 72 from 1 patients.
the value I am sending before encryption is : 11
the encrypted value was sent to : 8440
the value I am sending before encryption is : 48
the encrypted value was sent to : 8442
Connection from ('127.0.0.1', 51208)
Received decrypted value: 46
Total sum: 118 from 2 patients.
Sending the current aggregated value: 118 to the hospital
the value I am sending before encryption is : 118
the encrypted value was sent to : 8443

```

Figure 2: Log of Bob

```

[Tue Oct 22 13:45:19]-[0]-[ il_terminale_di_Lambione -> ~/ALL/ITU/SECURITY/HAI
python3 client_Charlie.py
Server listening on port 8440...
My public key has just been sent to : 8443
now processing the certificate from : Hospital@gmail.com
My public key has just been sent to : 8442
now processing the certificate from : Alice@gmail.com
now processing the certificate from : Bob@gmail.com
SENDING DATA FROM CHARLIE
the split generated the following -> 7 46 47 which adds up to the input : 100
My public key has just been sent to : 8441
Connection from ('127.0.0.1', 57224)
Received decrypted value: 4
Total sum: 11 from 1 patients.
Connection from ('127.0.0.1', 58066)
Received decrypted value: 11
Total sum: 22 from 2 patients.
sending the current aggregated value: 22 to the hospital
the value I am sending before encryption is : 22
the encrypted value was sent to : 8443
the value I am sending before encryption is : 46
the encrypted value was sent to : 8441
the value I am sending before encryption is : 47
the encrypted value was sent to : 8442

```

Figure 3: Log of Charlie

```
[Tue Oct 22 13:45:30]-[0]-[ il_terminale_di_Lambion
->python3 hospital_server.py
Server listening on port 8443...
processing certificate from : Alice@gmail.com
processing certificate from : Bob@gmail.com
processing certificate from : Charlie@gmail.com
sending pk to 8442
My public key has just been sent to : 8442
sending pk to 8441
My public key has just been sent to : 8441
sending pk to 8440
My public key has just been sent to : 8440
Connection from ('127.0.0.1', 45090)
Received decrypted value: 22
Total sum: 22 from 1 patients.
Connection from ('127.0.0.1', 45098)
Received decrypted value: 118
Total sum: 140 from 2 patients.
Connection from ('127.0.0.1', 45102)
Received decrypted value: 160
Total sum: 300 from 3 patients.
Final aggregated value: 300
```

Figure 4: Log of the Hospital

As the Hospital log reports, the aggregated value received was the right one 300 so the mission was accomplished through secure channels. Feel free to change the `patient_value` variable at the beginning of the clients file to see that the implementation is correct.