Question 1:

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?

Regarding GDPR:

Briefly looking at the scenario where we store a name and some value about that name, GDPR would only be breached if the data were enough to identify who the person is. Therefore, removing the name would reduce the risk of direct identification, as some value could be anything (and about anyone). However, removing the name alone is not enough, as other data points could still be used to re-identify the individual, especially if combined with external information.

Given that we are dealing with medical data, some value could potentially be a rare disease or something else that could make the person easily identifiable. In the case where we are storing a value from an experiment, it would generally be harder to identify the person in question just from that value alone. However, the risk still depends on the context of the value and whether it could be linked to other identifiable information.

Remaining risks in using Federated Learning with Secure Aggregation:

- Byzantine Attacks: Where we have some of the clients sending incorrect data to the server, which in our case degrades the performance of our model.
- Malicious Data: Adversaries that inject malicious data into the training process of our model to make the model incorrect. This can lead to incorrect predictions, which can be very dangerous when working with something like healthcare.
- Data Leakage: When sending the shares to the server, it can potentially leak information about the data. Adversaries might then be able to infer sensitive information by analyzing the sent shares, especially if they gain access to lots of shares sent over time.

Question 2:

Describe the adversarial model you are working on (1), describing the building blocks of your proposed solution, how they are combined in your final solution (2) and why they guarantee security against the adversary you describe (3).

(1) Adversarial model being worked on:

We are looking at a passive adversary. Passive adversaries primarily focus on gathering information compared to an active adversary that would try to bring down a system. You could say that passive adversaries are purely educational, and passive attacks could be used to prepare for a much larger active attack.

Also a Dolev-Yao attack is possible, where someone can intercept, modify or inject packets sent over the internet.

(2) Describe the buildings blocks of your proposed solution and how they are combined in the solution:

I have used TLS and Secret Sharing. Secret Sharing is used when the data has to be transferred between the clients and when sending the data to the server. Secret Sharing helps with not revealing what the value of a person actually is - unless the person has been compromised. In the end the server gets shares sent from the clients that in the end sums up to the aggregated value of the clients.

TLS is used to secure the connections between clients and between clients and the server. TLS provides authentication of the receiver, encryption of data sent through the connection, and ensures data integrity.

These techniques are combined in the solution by ensuring that every connection is secured with TLS, and the values sent through these connections are protected using Secret Sharing.

(3) How does the final solution guarantee security against the adversary described?:

By using Secret Sharing clients only distribute partial, unusable pieces of their *secret value*. Having a passive adversary that cannot gather enough of these pieces will not be able to learn what the original *secret value*.

TLS secures us against Dolev-Yao attacks because TLS authenticates, encrypts and guarantees data integrity. It does this by first having a handshake phase, where the client has to trust the server it tries to connect to. It does this by getting the TLS certificate from the server and verifies it using the public key-protocol. When this is verified, a **shared secret key** will then be established by server and client, and this **shared secret key** will be used to encrypt and decrypt data sent.