Byzantine Fault Tolerant Banking

- Highly Dependable Systems 2021/2022 -

Special Epoch

Marta Xavier 90752

Instituto Superior Técnico
MSc. Engineering and Computer Science

2. Project Architecture

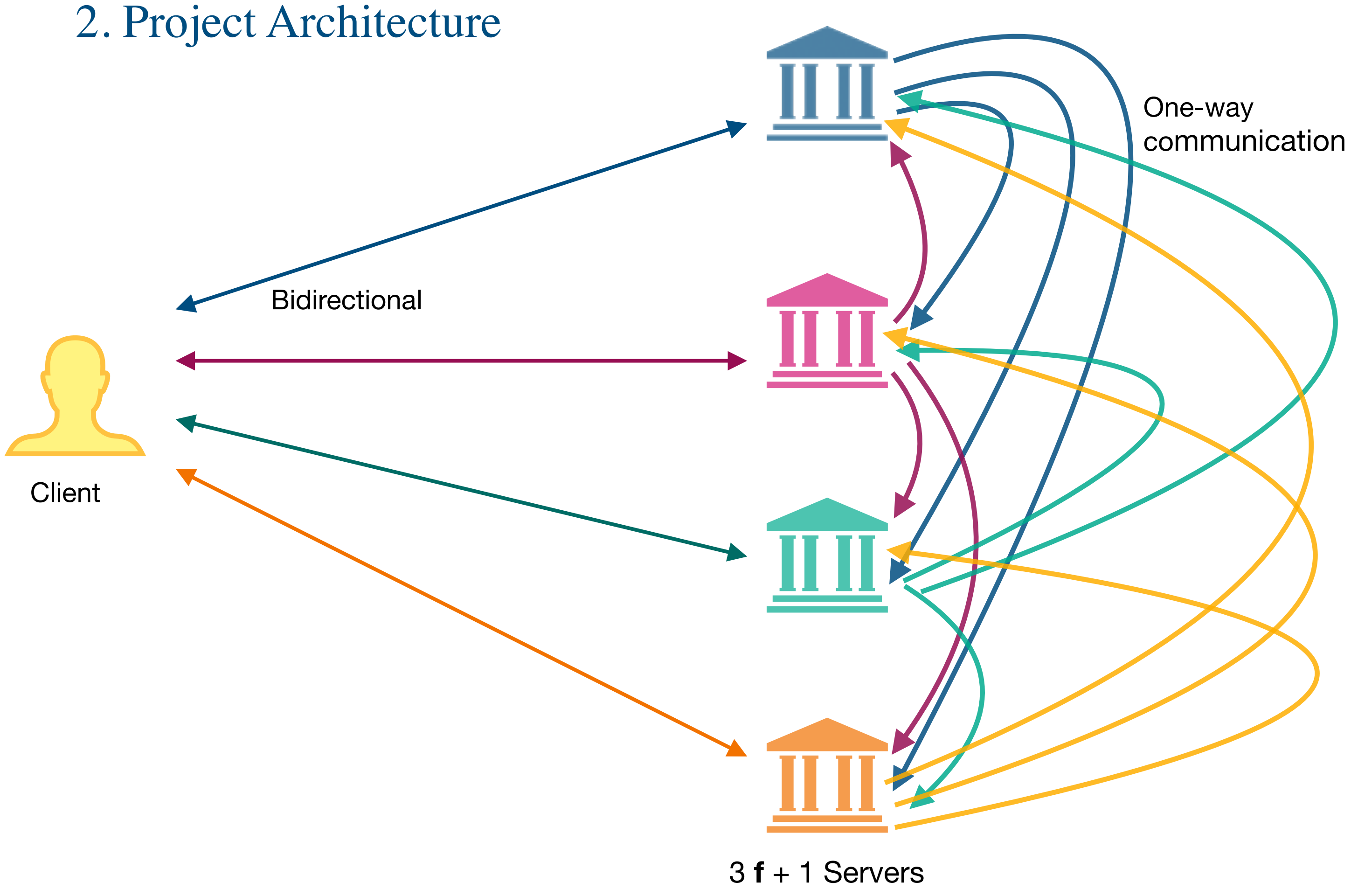
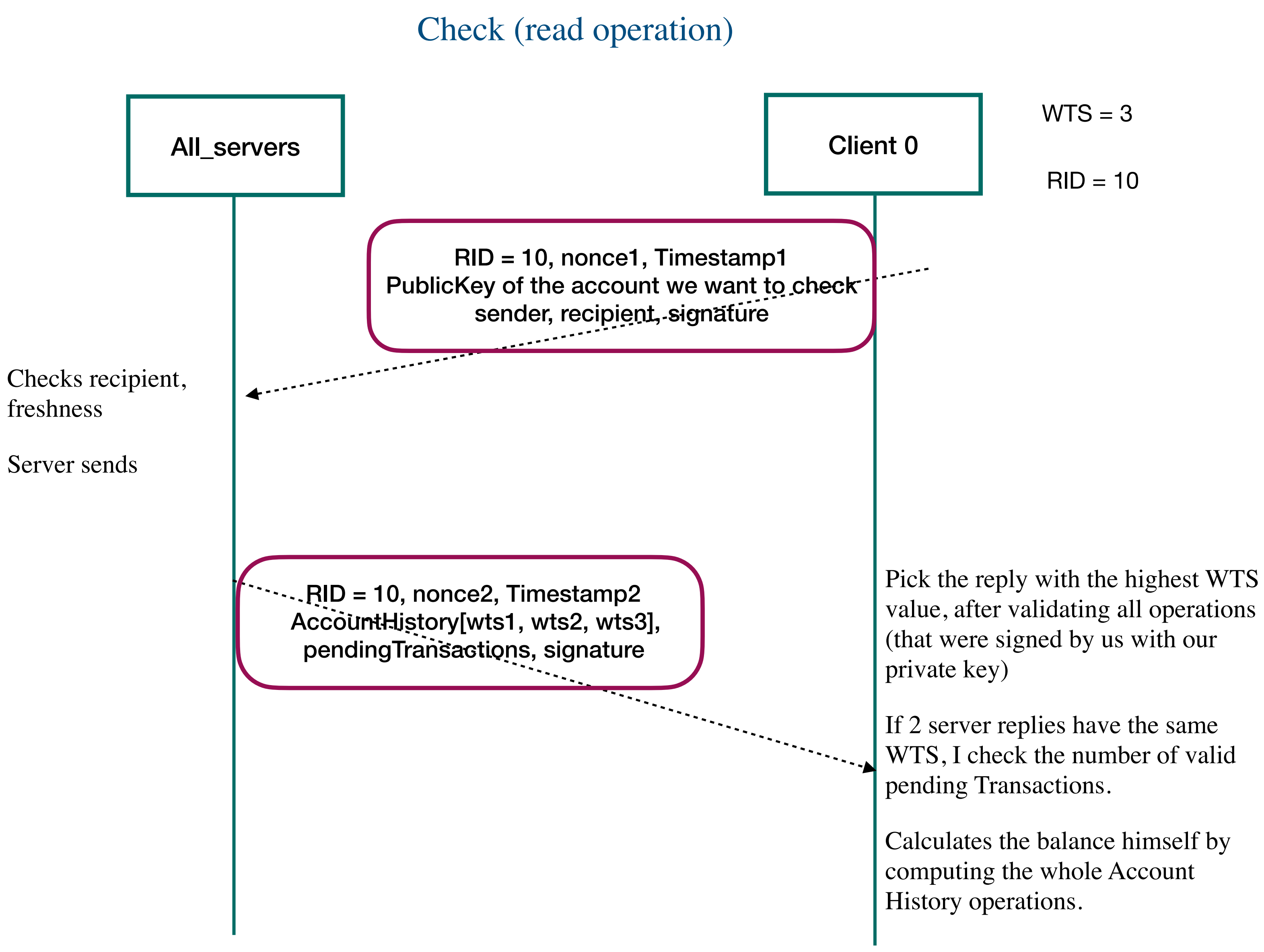


Fig1 - The system's channel architecture using **Authenticated Point-to-Point** links

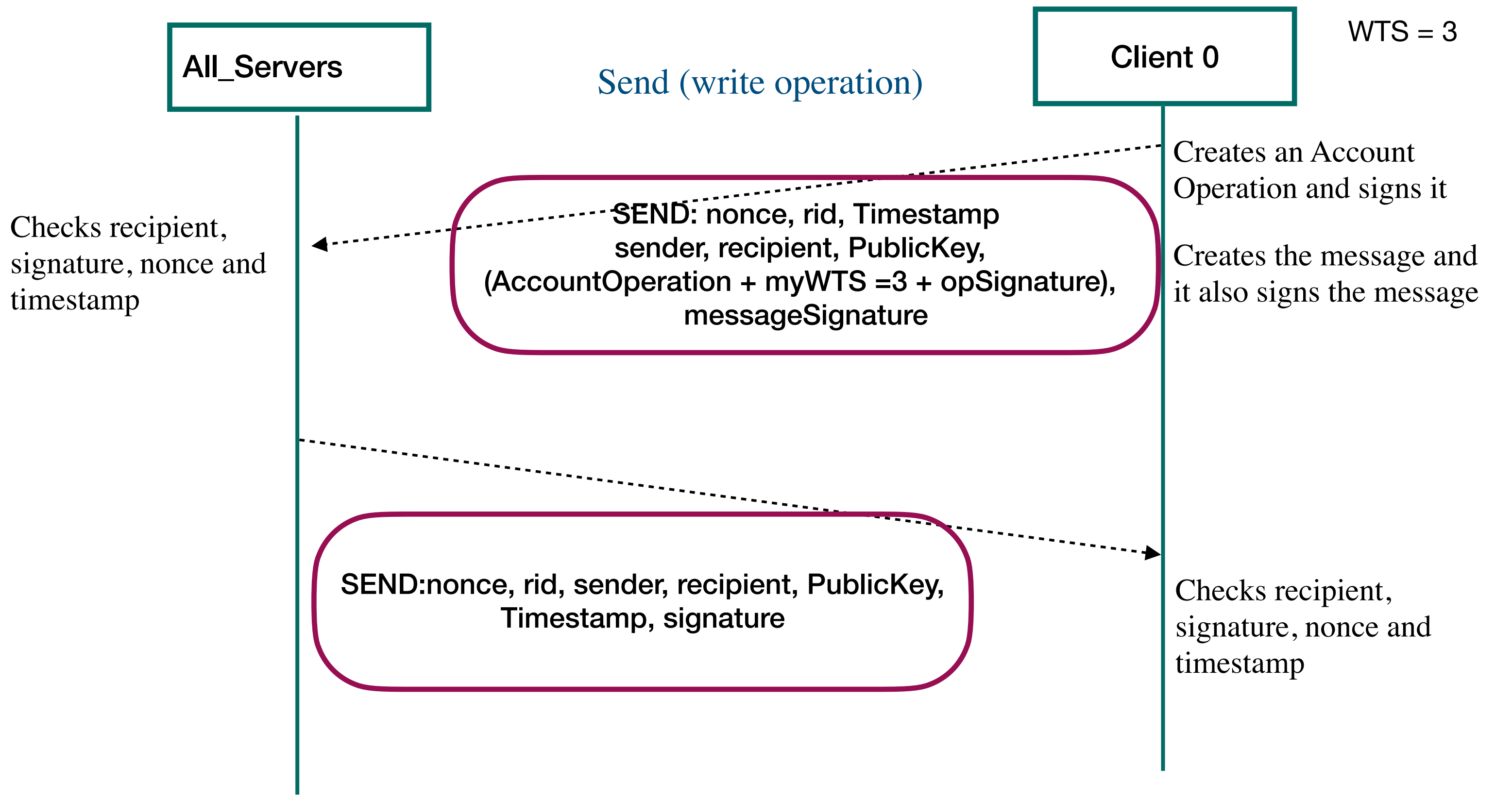
In this project, to tolerate **f** server faults, including Byzantine or Crash-faults, I implemented a modified version of the **(1, N) Byzantine Regular Register** on top of **Authenticated Point-to-Point links**.

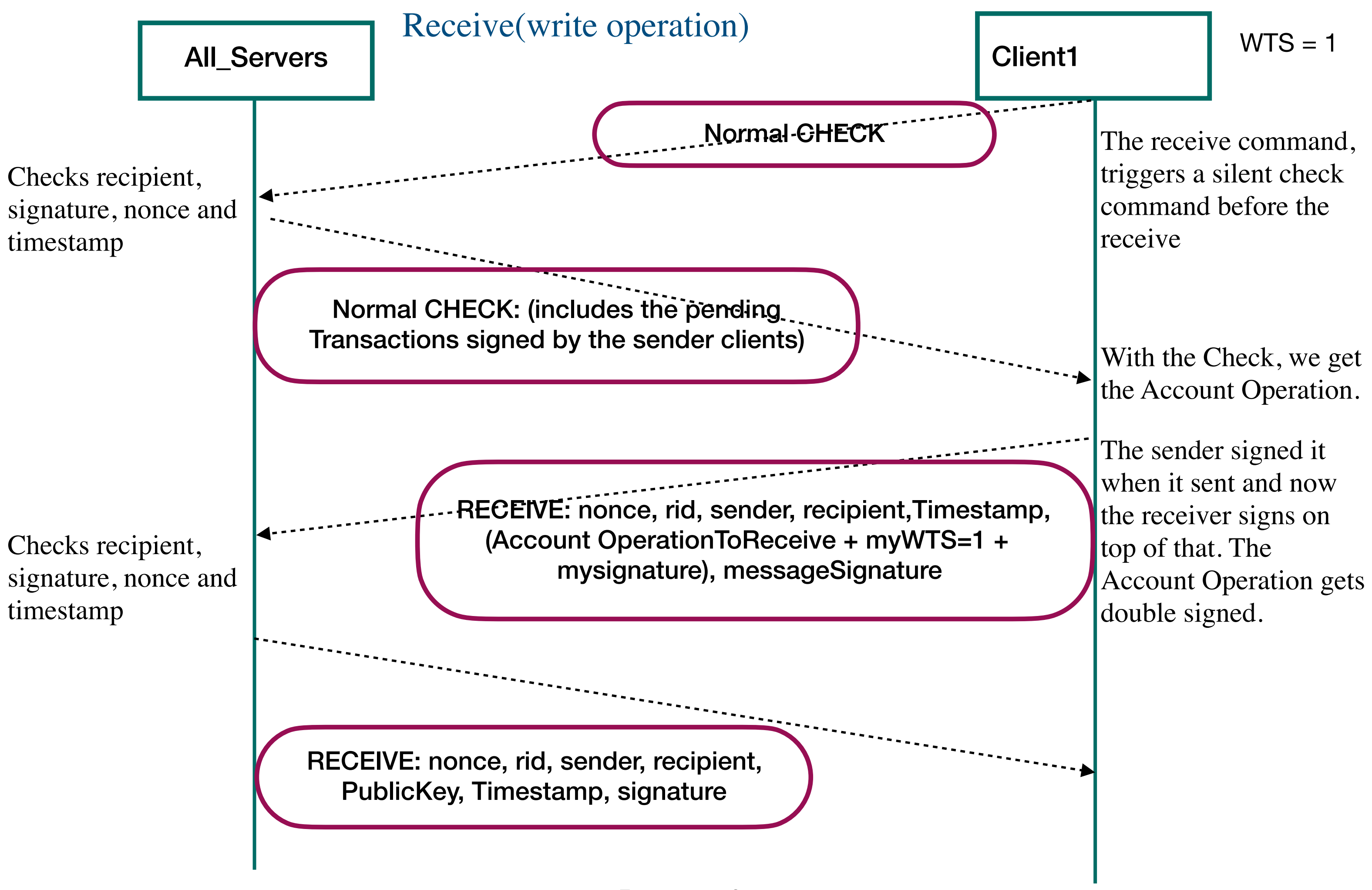
Clients initiate the communication by broadcasting a request to all servers. The server protects itself from **byzantine clients** that could try to send, for instance, different requests to different servers by **rebroadcasting** the client's request. For this purpose, we implement the **Authenticated Double Echo Broadcast** algorithm on the servers on top of **Authenticated Point-to-Point links**.

2.1 Protocol - Byzantine Regular Register


Check (read operation)
All_servers
Client 0
WTS = 3
RID = 10
RID = 10, nonce1, Timestamp1
PublicKey of the account we want to check
sender, recipient, signature
Checks recipient, freshness
Server sends
RID = 10, nonce2, Timestamp2
AccountHistory[wts1, wts2, wts3],
pendingTransactions, signature
Pick the reply with the highest WTS value, after validating all operations (that were signed by us with our private key)
If 2 server replies have the same WTS, I check the number of valid pending Transactions.
Calculates the balance himself by computing the whole Account History operations.

Upon performing **any** operations, the client increments a read timestamp (**RID**). When he performs **write** operations, he increments a write timestamp (**WTS**). He keeps both values stored **persistently.**


All_Servers
Send (write operation)
Client 0
WTS = 3
Creates an Account Operation and signs it
Creates the message and it also signs the message
SEND: nonce, rid, Timestamp sender, recipient, PublicKey, (AccountOperation + myWTS =3 + opSignature), messageSignature
Checks recipient, signature, nonce and timestamp
SEND:nonce, rid, sender, recipient, PublicKey, Timestamp, signature
Checks recipient, signature, nonce and timestamp


All_Servers
Receive(write operation)
Client1
WTS = 1
Normal CHECK
The receive command, triggers a silent check command before the receive
Checks recipient, signature, nonce and timestamp
Normal CHECK: (includes the pending Transactions signed by the sender clients)
With the Check, we get the Account Operation.
The sender signed it when it sent and now the receiver signs on top of that. The Account Operation gets double signed.
RECEIVE: nonce, rid, sender, recipient,Timestamp, (Account OperationToReceive + myWTS=1 + mysignature), messageSignature
Checks recipient, signature, nonce and timestamp
RECEIVE: nonce, rid, sender, recipient, PublicKey, Timestamp, signature

Authenticated Double Echo Broadcast

RebroadcastMap	**Echo**Map	**Ready**Map
<nonce0, List rebroadcast>	<nonce0, List echo>	<nonce0, List ready>
<nonce1, List rebroadcast>	<nonce1, List echo>	<nonce1, List ready>

This implementation follows the 3stages - Rebroadcast, Echo, Ready.

In each stage, it guarantees **Authenticated Point-To-Point Perfect Links**, checking if the messages are **valid** and **non-repeated.**

It also verifies the Rebroadcast, Echo, and Ready messages originated from a **server** and that the **original piggybacked** messages originated from a **client**.

3. Threat Analysis and Protection

Digital signature - I obtain the digital signature by calculating the hash of the message and encrypting its value with the private key. Anyone in possession of the respective public key can verify the signature.

Freshness - To ensure freshness, a nonce and a timestamp are sent alongside the data and are included in the information the digital signature depends on so that I can detect any attempt to tamper with them. The timestamp is important so that I can discard nonces that are too old.

Non-repudiation - The assurance that an entity is not able to deny having sent or received a message. This property is guaranteed in all messages exchanged between the client and the server through the use of digital signatures.

Replay attacks - By adding a freshness token to the messages that traverse the network, it is possible to prevent the unauthorised replay of information. A malicious entity can still intercept a valid message and can try to relay it, however, the system's application-level security will assess the validity of the timestamp and the uniqueness of the nonce. To disregard previously seen messages. To prevent Replay attacks that involve sending that reply to different entities, are avoided by signing the destination of the request inside the message.

Sybil attacks - is a multiple identity problem. An assumption made for this project is that there is a Public Key Infrastructure in place that would be responsible to assign each entity of the system its own private key and to assure the server has knowledge of this association. Taking this assumption into consideration, sybil attacks don't apply to this project.

Man in the Middle attacks usually exploit unauthorised insertion of information by an attacker that positions himself between the client and the server. These threats are avoided by adding a digital signature (prevents modifications) and freshness tokens including a timestamp (shortens the window of the attack) and a nonce (invalidates duplicated replies) to all messages exchanged, thus providing integrity and authentication, allowing the detection of modifications made to the original message and discarding any unauthorised request/response.

Specific Man in the Middle attack - An example that is prevented by the implementation of RID and WTS to work as a challenge response on top of nonces, timestamps, and message recipients, as these three would not suffice. After receiving 2f + 1 valid replies from the servers, the client ignores the remaining replies. A man in the middle could store the ignored replies (would have nonces unknown to the client, timestamps would only shorten the window of the attack and the message recipient would be correct) and after having enough messages he could replay them. However, the RID and WTS values would signal the client to an older message.

Dictionary attacks - The private keys are stored in a file, encrypted with a **password** and a **salt** - the implementation was not completed and it can be found on “encryptAndStorePrivKey”

4. Dependability

By replicating the server **N = 3f + 1**, we ensure that even with **f** server faults we can still ensure the **availability** of the system

I use **Point To Point Authenticated Perfect Links** as we sign every message (with nonce, timestamp, and intended recipient) that goes into the communication TCP channel, ensuring **no duplication** and **no creation**

Ensure the **persistence of data** even in case of crash-faults by having the client and the server store their state in a persistent file. To protect against corrupted or incomplete storage, I create a temporary logger file and **atomically** rename it to replace the original logger.

5. Weaknesses

- Service Overloading (**DOS**). This attack is not being dealt with in this project and this could be implemented by requiring the client to do some computation (**proof of work**) alongside heavy requests or requests requiring broadcast

- **Amplification attacks** - Since this system is very **broadcast-heavy** to deal with byzantine clients, the system is vulnerable to amplification attacks

- **Client-server collusion**