Correction over the first stage

Passphrases on Keys

We created the Virtual Keys used on the project and ciphered them with a passphrase to guarantee that if someone gets hold of the key that person will not be able to use it to forge the owner's identity. To facilitate testing and demonstration of the project all passphrases are stored on a static array that whenever a passphrase is needed it can be automatically retrieved from there.

Atomic files

On the previous delivery the files used to keep notary state could be corrupted if it crashes during a write. At this stage we fixed that issue by using a temporary file to handle all the writes, and only if all the write succeeds the swap between temporary and main file is performed.

Anti-Spam Mechanism

To guarantee that a malicious user cannot overload the server with a big number of requests we implemented a Proof of Work. This Proof of Work consists on calculating a number that when added to the message that the client wants to send the calculated hash has the first 3 bytes equal to zero. This operation shows that the user used a significant amount of CPU resources for us to believe that the request is a real request and not a DoS attempt.

Byzantine clients

A byzantine client can try to corrupt the consistency between the notaries, by not delivering a request to some notaries or sending different requests to different notaries.

Our implementation handles all these issues. Notaries before executing any write operation, perform the algorithm ***Authenticated Double-Echo Broadcast***, but with some modifications, namely:

- Digital signatures off all *"echo"* and *"ready"* messages, and logical clocks to prevent resending of any of these messages. The clocks of these messages are persistently stored in *"RBClocks"* file, with this we never accept a message from the past, even if the notary crashes and then goes up.
- In the pseudo code of the algorithm, we are supposed to wait for a quorum of "*echo"* messages equals to the one that we send and for $2f + 1$ *"ready"* messages. We changed that to wait for a quorum number of equal messages for the *"echo"* phase and to wait for $2f + 1$ equal messages in the *"ready"* phase.

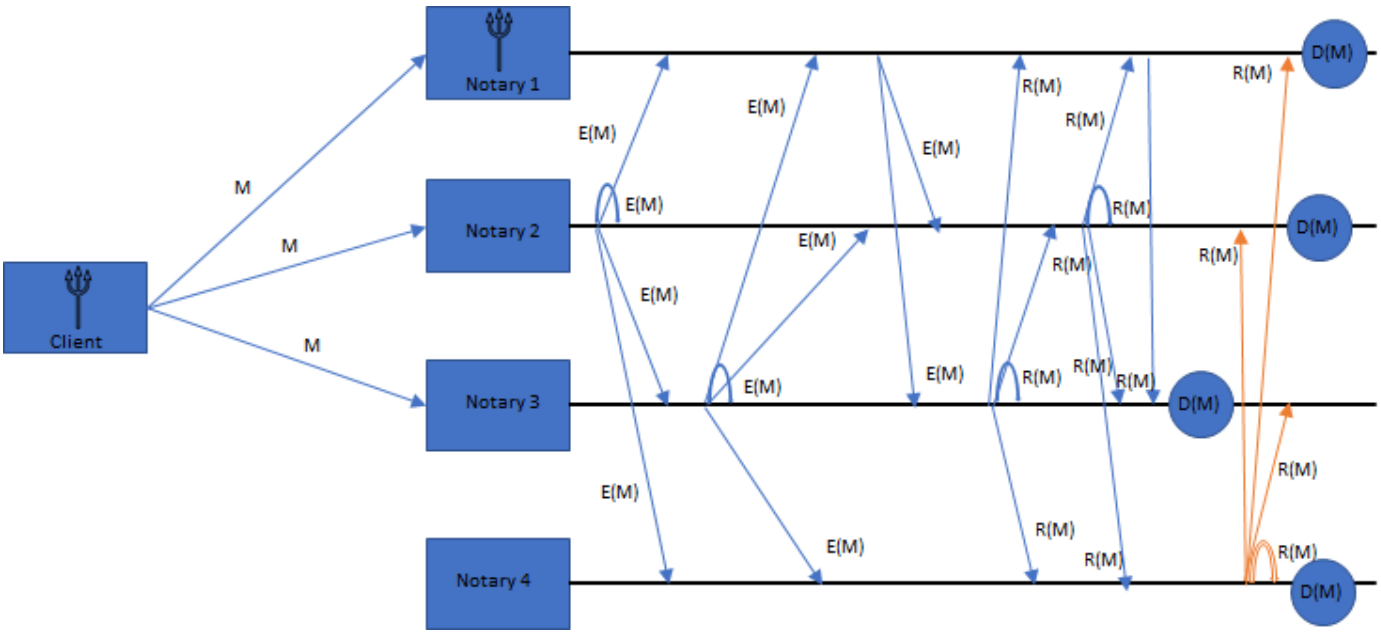
With these adaptations over the base algorithm we never reach an inconsistent state between notaries. Over the assumption that we only have f number of faults, we guarantee that, or all the notaries perform the operation in the exact same way, or the operation is not performed at none of the notaries.

Byzantine notaries

Since all "echo" and "ready" messages are signed and have logical clocks to prevent replay attacks, we can prevent the case where notary tries to resend a message or to send a forged one. The remaining case is the one that the byzantine notary can send messages to some notaries and does not send to others, the reliable broadcast algorithm used prevents that this scenario leads to an erroneous state over the notaries.

Our implementation also supports the case where we have a byzantine notary and a byzantine client trying to break the global consistency of the system.

Consider the following scenario:

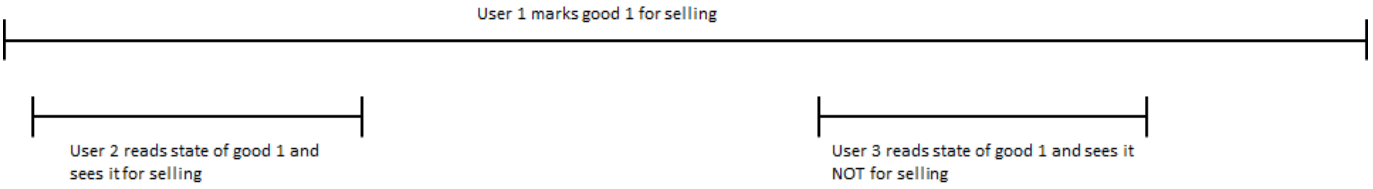


Our algorithm performs an extra phase, named *"amplification phase"*. This phase allows the notary who did not received the request from client, receive the request from the other non Byzantines notaries and execute the operation, even in the presence of a simultaneous byzantine notary, the consistency throughout the notaries does not break.

Byzantine regular register and extension to atomic

We started by implementing a byzantine regular register in the client side. With this algorithm we assured that without any concurrent write, read always return the most recent value written. If the write is happening, then we have no guarantee about the value read.

Consider the following scenario:



Since regular register does not guarantee that the read value is the most recent one while a write is occurring, some problems may arise. In the scenario presented above, user 3 receives an old value comparing with user 2.

To solve this kind of issues we extended the byzantine regular register algorithm to the byzantine atomic register one.

The atomic guarantees that if a read returns a value then all subsequent reads will return. This scenario is impossible to happen over the atomic register implementation.

Using atomic register, if user 2 reads that the item is for selling, then user 3 will also see it. This is achieved with the introduction of a new phase, named “write-back” phase. In this phase when a user tries to read a state of a good, it waits for a quorum of responses, then performs the write of the most recent value received, then it waits for a quorum of ACKs and finally it considers the read as finished. This way the next read will for sure always return the most recent value.