Instituto Superior Técnico

==

SEC Report 2

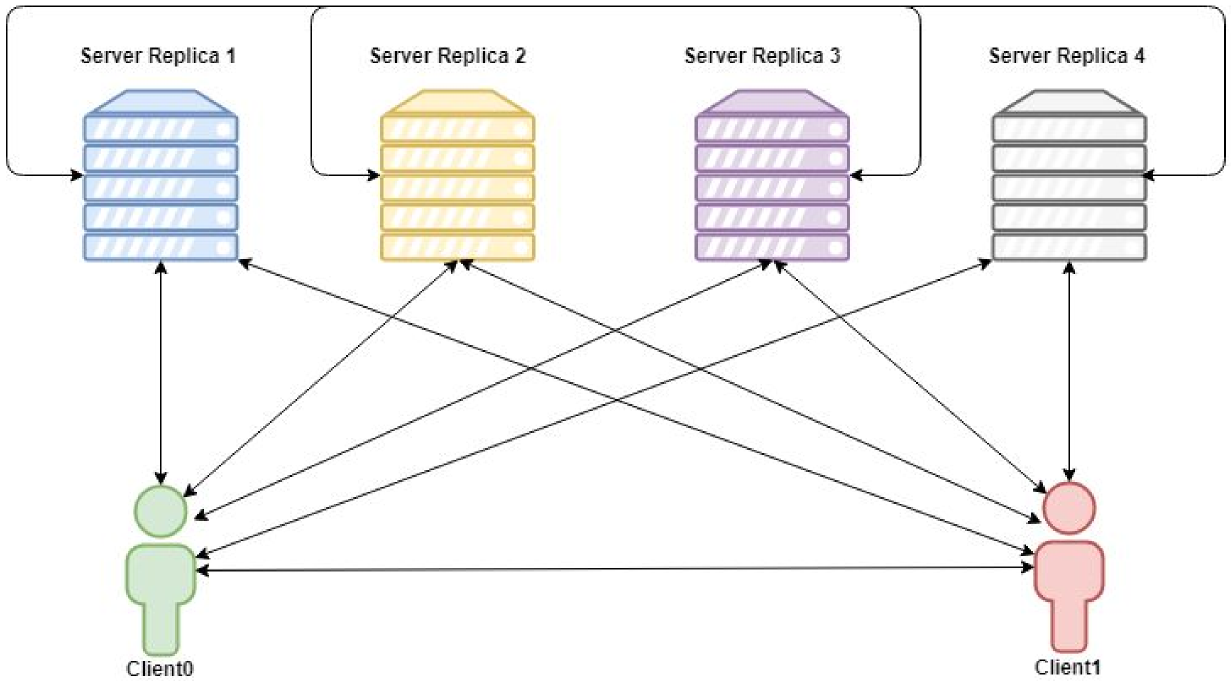
==

Group 23

André Fonseca 84698
Leonor Loureiro 84736
Sebastião Amaro 84767

Professor: Nuno Anselmo
Date: 16/05/2019

Design



To be able to tolerate faults (crash and byzantine), the system contains several server replicas. The number of replicas (**N**) depends on the number of faults (**f**) to be tolerated. This system requires **N** = 3**f** + 1 replicas.

To control the access to replicas, a **Modified** version of **(1, N) Byzantine Atomic Register** was implemented, mapping each good to a register. This algorithm is the bottleneck determining the need for 3f+1 Replicas.

(1, N) Byzantine Atomic Register Implementation:

- **Read** operations:
  - GetStateOfGood
- **Write** Operations:
  - Intention To Sell
  - Transfer Good
- A **(1, N) regular register** is adapted to **(1, N) byzantine regular register**
  - The links are replaced with **authenticated perfect links**
  - The **quorum majority of** 3f + 1 instead of 2+1
  - The Clients sign the **good's id**, **timestamp**, **value**, **good owner**'s id and **writer**'s id, maintained by the register
- The **(1, N) byzantine regular register** adapted to a **(1, N) byzantine atomic register**:
  - Added a writeback phase to read operations

The **(1, N) byzantine Atomic Register** does not allow **different writers** nor **byzantine clients**, requiring the following **modifications**:

- Before writing to a good, the client must first request the current timestamp of the good. This modification is sufficient because the server only allows write operations from the current owner of the good, at each point in time.
- Instead of a **best effort broadcast**, use a **reliable byzantine broadcast**, which allows the sender to be faulty.
    - The Servers have to communicate between themselves.

Authenticated Perfect Links Implementation:

- Each arriving message's **authentication** is **verified** with the sender's public key
- Each message before being sent is **authenticated** with the sender's **signature**
- **Modification**:
    - Every message contains a unique identifier, a **nonce**, that makes every message unique and allows for a freshness verification.

Reliable Byzantine Broadcast Implementation:

- Follows the **Double Echo Authenticated Broadcast** algorithm
    - Uses **Authenticated Perfect Links**
    - Tolerates faulty clients while maintaining reliability.
- **Modifications:**
    - Once a **write request** starts on a certain **good**, every other request that attempts to write on the same good is **denied** until the write request is processed and delivered. Provides a **safety** to writes, ensuring server consistency.

Proof of work implementation:

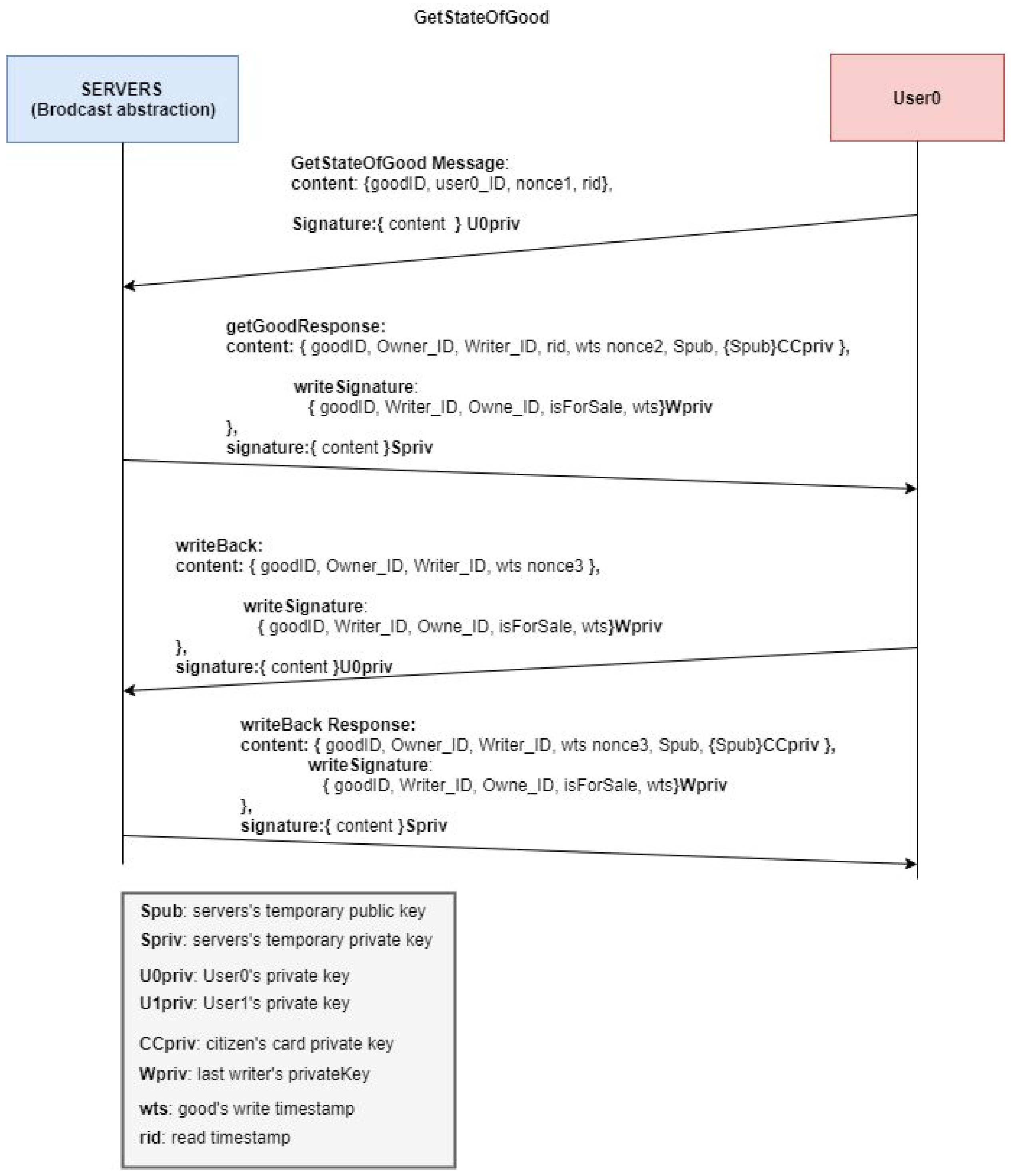
- In order for the TransferGood message to be approved, it has to contain a proof of work.
    - The client must find the integer that will result in the hash of the message starting with a certain prefix.

<u>Assumptions</u>:

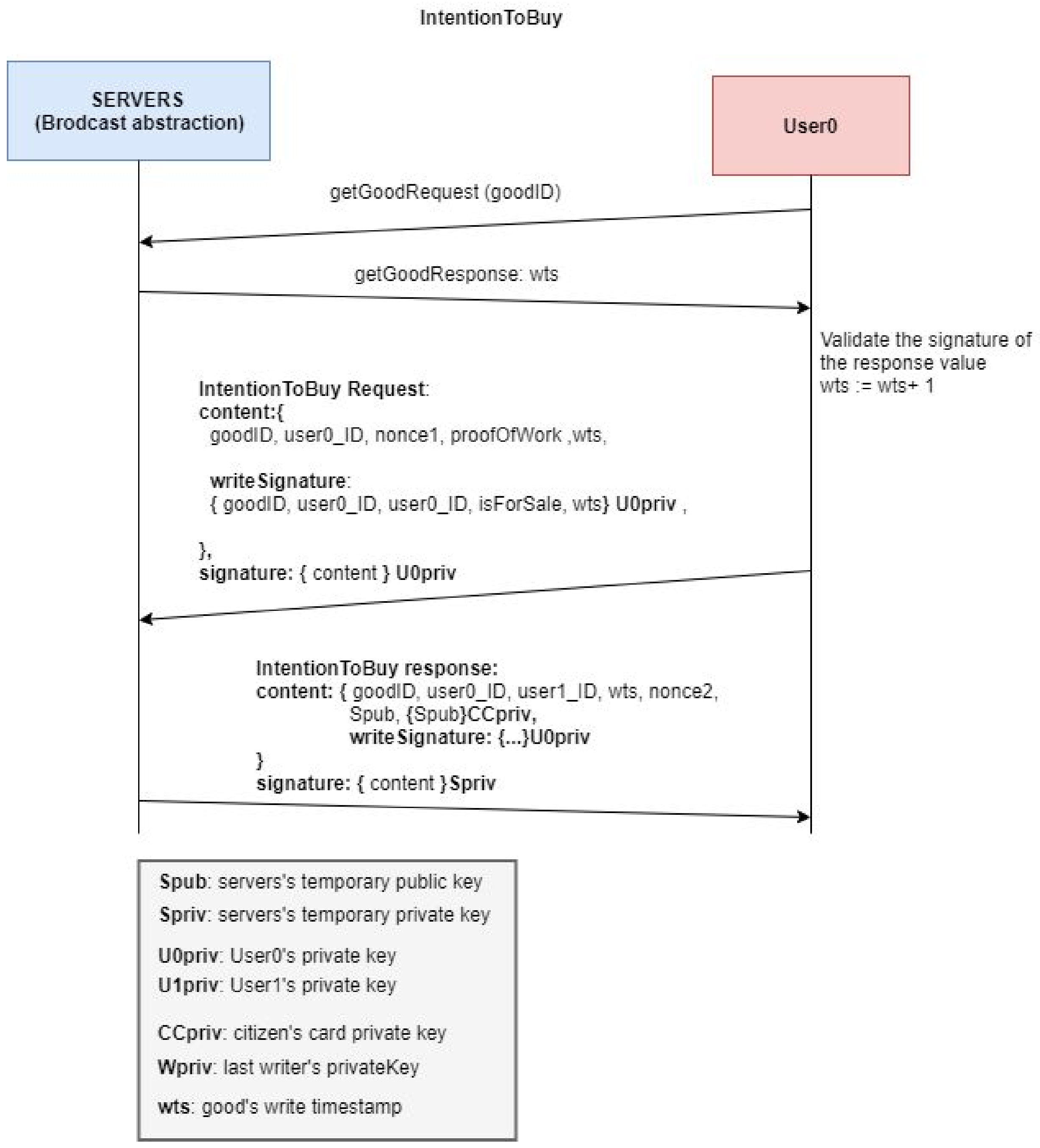
Every client and server in this system has a key pair and know everyone else's public keys.

Protocol:

GetState Of Good


GetStateOfGood
SERVERS (Brodcast abstraction)
User0
GetStateOfGood Message:
content: {goodID, user0_ID, nonce1, rid},
Signature:{ content } **U0priv**
getGoodResponse:
content: { goodID, Owner_ID, Writer_ID, rid, wts nonce2, Spub, {Spub}**CCpriv** },
write Signature:
{ goodID, Writer_ID, Owne_ID, isForSale, wts}**Wpriv**
},
signature:{ content }**Spriv**
writeBack:
content: { goodID, Owner_ID, Writer_ID, wts nonce3 },
write Signature:
{ goodID, Writer_ID, Owne_ID, isForSale, wts}**Wpriv**
},
signature:{ content }**U0priv**
writeBack Response:
content: { goodID, Owner_ID, Writer_ID, wts nonce3, Spub, {Spub}**CCpriv** },
write Signature:
{ goodID, Writer_ID, Owne_ID, isForSale, wts}**Wpriv**
},
signature:{ content }**Spriv**
Spub: servers's temporary public key
Spriv: servers's temporary private key
U0priv: User0's private key
U1priv: User1's private key
CCpriv: citizen's card private key
Wpriv: last writer's privateKey
wts: good's write timestamp
rid: read timestamp

Intention To Sell



IntentionToBuy Request:
content:{
goodID, user0_ID, nonce1, proofOfWork ,wts,

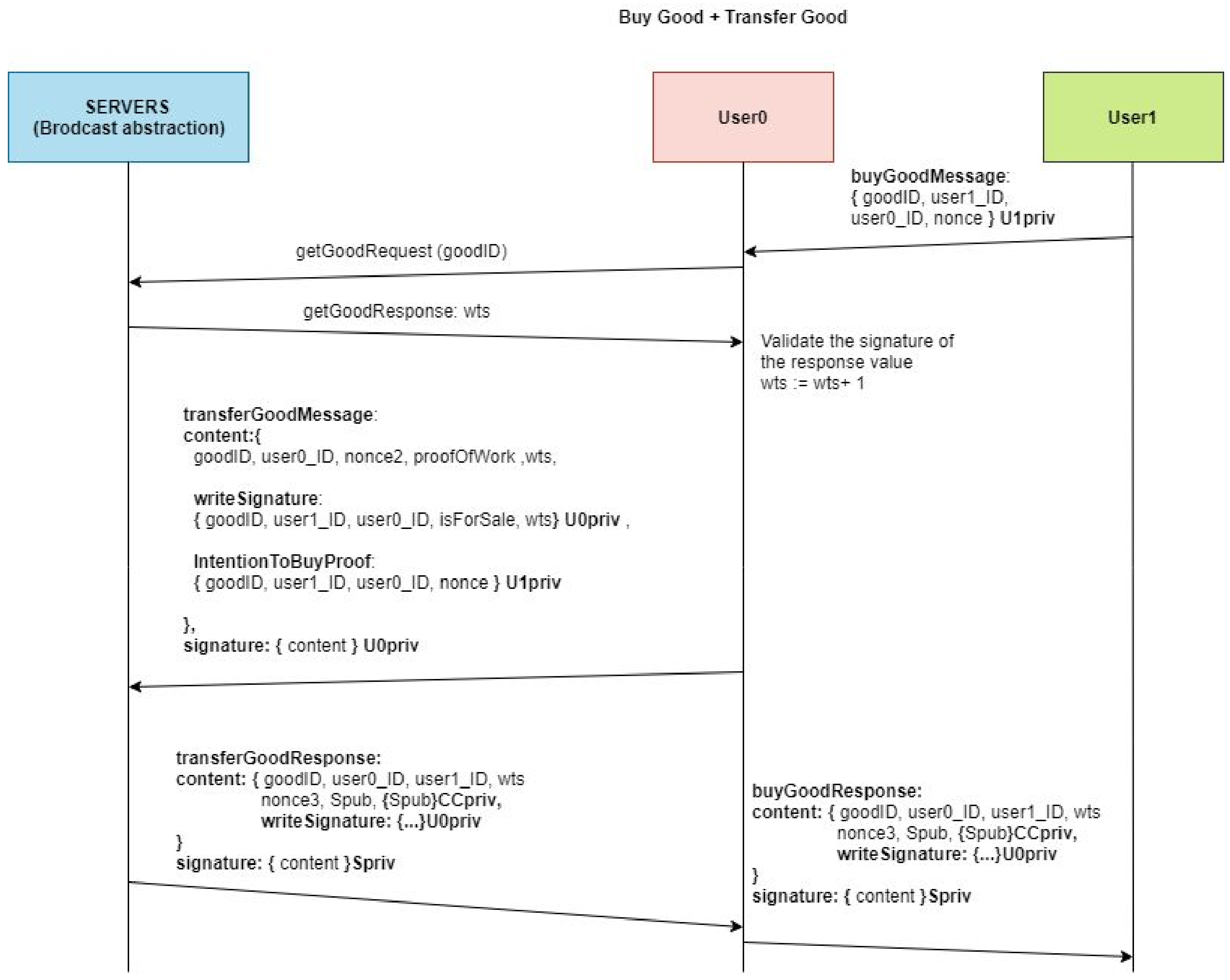
write Signature:
{ goodID, user0_ID, user0_ID, isForSale, wts} **U0priv** ,

},
signature: { content } **U0priv**

IntentionToBuy response:
content: { goodID, user0_ID, user1_ID, wts, nonce2,
Spub, {Spub}**CCpriv,**
write Signature: {...}U0priv
}
signature: { content } **Spriv**

Spub: servers's temporary public key
Spriv: servers's temporary private key
U0priv: User0's private key
U1priv: User1's private key
CCpriv: citizen's card private key
Wpriv: last writer's privateKey
wts: good's write timestamp

TransferGood


Buy Good + Transfer Good
SERVERS (Brodcast abstraction)
User0
User1
buyGoodMessage: { goodID, user1_ID, user0_ID, nonce } U1priv
getGoodRequest (goodID)
getGoodResponse: wts
Validate the signature of the response value
wts := wts+ 1
transferGoodMessage:
content:{
goodID, user0_ID, nonce2, proofOfWork ,wts,
writeSignature:
{ goodID, user1_ID, user0_ID, isForSale, wts} U0priv ,
IntentionToBuyProof:
{ goodID, user1_ID, user0_ID, nonce } U1priv
},
signature: { content } U0priv
transferGoodResponse:
content: { goodID, user0_ID, user1_ID, wts
nonce3, Spub, {Spub}CCpriv,
writeSignature: {...}U0priv
}
signature: { content }Spriv
buyGoodResponse:
content: { goodID, user0_ID, user1_ID, wts
nonce3, Spub, {Spub}CCpriv,
writeSignature: {...}U0priv
}
signature: { content }Spriv

Spriv: servers's temporary private key
U0priv: User0's private key
U1priv: User1's private key

Spub: servers's temporary public key
CCpriv: citizen's card private key

wts: good's write timestamp

Properties

Security:
Integrity and Non-repudiation + Authentication
- Provided by the Authenticated Perfect Links, through the use of digital signatures
- (1, N) Byzantine Register, all writes are signed by the writer.

Freshness
- Provided by the Authenticated Perfect Links modification to include a nonce in each message.

Dependability:
The server and client state is **persistent**, updates are all performed **atomically**.
Availability, the servers are **replicated**, requiring only a correct majority to proceed.
- Resistant to byzantine faults

Spam Combat , client's computational Investment through the mandatory **proof of work**.

Communication(Authenticated Perfect Links + Reliable Byzantine Broadcast):
Validity: If a correct process p broadcasts a msg m, then every correct process eventually delivers m.
No duplication: no message is delivered more than once to the same process
No creation: no message is delivered unless it was sent
Integrity: If some correct process delivers a message m with sender p and process p is correct, then m was previously broadcast by p
Consistency: If some correct process delivers a message m and another correct process delivers a message m′, then m = m′.

Possible threats:

Unauthorized insertion of forged information and modification of information in transit (Man in the middle)
- Prevented by the properties **Integrity and Non-repudiation + Authentication** of the Authenticated perfect links.

Unauthorized replay of information (replay attacks)
- Prevented by the **freshness** property.

Service Overloading (DOS and DDOs)
- These attacks are diminished on **transfer good operation**, that requires the client's computational investment, the **proof of work**.

Malicious Client

- The server is protected from byzantine clients by implementing the reliable byzantine broadcast algorithm.
- Spam attacks are prevented by requiring that the client present the **proof of work** mentioned above.

Quality of Life Improvements and Optimizations

The server uses the notary's Citizen Card to **sign** a **temporary key** pair which is used for signing instead of the CC's key, allowing the Citizen's Card pin to be input only at the start of the server. This optimization also diminishes the chance of the CC's key being broken because it allows the CC's key to be used less often.