A Decentralized Public Key Infrastructure with Identity Retention

Marco Pesic
marco.pesic@studio.unibo.it

Riccardo Alni
riccardo.alni@studio.unibo.it

January 9, 2025

Public Key Infrastructures (PKIs) enable users to look up and verify one another's public keys based on identities. One of the biggest vulnerabilities in current approaches to PKIs is the lack of identity retention, meaning a user can register a public key under another's already-registered identity. This project aims to design and implement a PKI that ensures this security property utilizing the consistency guarantees provided by cryptocurrencies. Our primary objective is to conduct an in-depth analysis on the blockchain in order to proper understand how to build one and what benefits this technology might bring in terms of security.

1 Goals/requirements

The goal of this project is to build a Public Key Infrastructure (PKI) on top of a simplified version of a blockchain based on the UTXO model. Every user is able to perform a small set of simple operations implemented on the PKI:

- Register an identity (e.g. a domain) with a corresponding public key
- Update the public key corresponding to a previously-registered identity
- Look up a public key corresponding to a given identity
- Revoke the public key corresponding to an identity

This project allows us to study in more depth the inner design of the blockchain technology, the problems with its implementation and the advantages in terms of robustness.

1.1 Scenarios

Currently, the most commonly employed approaches to public key infrastructures fall into two categories: Certificate Authorities (CAs) and decentralized networks of peer-to-peer certification, also referred to as Webs of Trust. The inability of both approaches to reliably offer identity retention is cause of inconsistency.

A Certificate Authority acts a trusted third party that is responsible for distributing and managing digital certificates for a network of users.

A Pretty Good Privacy (PGP) Web of Trust system eliminates the problem of having a single point of trust since authentication is entirely decentralized: users are able to designate others as trustworthy by signing their public keys.

Our solution offers a completely decentralized PKI that leverages the consistency offered by blockchain-based cryptocurrencies such as Bitcoin to provide a stronger identity retention guarantee than any of the PKIs used in practice. Instead of having to trust a third party as in the CA system or a small set of users as in the PGP system, it only requires that users trust that the majority of other users is not malicious.

Therefore, the most common use case is if Alice is communicating with Bob, how can she leverage the infrastructure to prove to Bob that she is actually Alice? The process does not differ from any other PKI. Alice sends Bob a message containing her identity and her online public key. Bob verifies that the public key does indeed correspond to the identity and then he sends Alice a random challenge message h. If Alice can respond with a produced digital signature (done on the message h with her secret key) such that the verification algorithm (using Alice's public key) returns true, Bob believes that she is the owner of that identity and therefore Alice has authenticated successfully.

2 Background

Our studies have been focused around Distributed Ledgers Technology (DLT) and Blockchain technology (BCT) since creating our own cryptocurrency is the main goal of this project. Further insights have been made on the PKI topic, mainly from the paper that we referenced as the starting point of the project [1].

2.1 Blockchain

Our implementation of a blockchain is very simple and it follows the structure we have also studied during the course. Bitcoin was our biggest inspiration as we chose to use the Unspent Transaction Output (UTXO) model and Proof Of Work (POW) as the consensus algorithm.

2.2 PKI and Double Signatures

A public key infrastructure is a set of processes, technologies and policies needed to encrypt and sign data in order to verify a user's identity by means of their public key.

It becomes necessary for those activities where simple passwords are an inadequate authentication method and more rigorous proof is required to confirm the identity of the parties involved in the communication and to validate the information being transferred. PKIs have two main security goals in addition to the classic CIA triad (Confidentiality, Integrity and Authenticity): accurate registration and identity retention. Accurate registration is the inability of a user to register an identity that does not belong to him; identity retention is the inability of a user to impersonate an identity already registered to someone else. The latter is probably the most desirable security property and the one we focused on achieving for this project, also because in some applications accurate registration may not be relevant at all. To offer even more security every identity is associated with two key pairs called online and offline. The online key pair is used to authenticate an identity by signatures while the offline key pair is stored securely offline so that it is not vulnerable to many of the threats that the online key might face and it is used to revoke old keys and sign new keys in case of a key compromise. This means that even if the user's online keys are stolen, if they can detect it in time they will be able to update their compomised key and avoid the risk of impersonification; revocation is necessary only if both key pairs are simultaneously compromised which is computationally complex.

2.3 Background on technologies and frameworks used

The whole project is developed in Java mainly for familiarity reasons. Amongst the many useful frameworks and libraries at our disposal, we opted for Bouncy Castle (for cryptography), RocksDB (for storage) and Kryo (for serialization).

3 Requirements Analysis

3.1 Functional requirements

The functional requirements of the application encompass all the features and services provided by the system:

- Every user can register a public key under a non-registered identity and update or revoke it
- Every user can look up the public key corresponding to his own identity
- Every user can access as a miner, having access to the PKI operations while also mining blocks to receive financial compensation.

3.2 Implicit requirements and assumptions

Implicit requirements, not explicitly defined above but nevertheless fundamental as they influence the design of the system, include:

- The underlying network is asynchronous: messages can be arbitrarily re-ordered or delayed, hence the system is vulnerable to byzantine faults
- We are counting on the nonexistence of a majority willing to subvert our cryptocurrency: we assume that more than half of the nodes are not faulty
- There is always at least one miner active on the blockchain
- We assume that the DNS node is always up so that new nodes when joining can exhange receive at least the list of neighbour nodes.

3.3 Non-functional requirements

Non-functional requirements express the characteristics and properties that a system must meet. The following non-functional requirements are present in the project:

- Security: while the project may be quite simple and not very secure network side, we focused on building an underlying architecture that is able to offer inherent fault tolerance, redundancy and transparency
- Extendability: at the moment the PKI only functions locally, but the network part of the system can be changed without altering its foundations.

3.4 Models / Paradigm / Technology

Due to time reasons, we opted not to implement a full network. Instead we built a similar offline version by means of a P2P communication working between local nodes via TCP.

3.5 Abstraction gap

An abstraction gap is present between the models and paradigms used and the problem to be solved: in reality nodes are distributed in a network, our implementation only simulates that behavior.

4 Design

The design of the project can be subdivided in 5 parts:

- Blockchain design: the essential building blocks for a functioning blockchain
- Sanity check design: a set of rules that allows for checking whether a block or a transaction is valid
- Network design: how nodes interact between each other
- Communication protocol design
- Node design: two types of nodes (Miners and Users) and their operations.

4.1 Blockchain design

Designing the blockchain was the key part of the project. We opted for a UTXO based blockchain with POW as consensus algorithm. Naturally we simplified the project by eliminating the design of the scripting language (like in bitcoin) because out of scope. Our key-delivery service is unauthenticated and thus susceptible to poisoning, routing and Sybil attacks. We could protect against poisoning and routing by requiring nodes to authenticate all of their communication using the cryptocurrency's authentication protocol. Authentication also discourages Sybil attacks, since there is some small cost associated with registering an identity and authentication requires that any node participating in the network hold a valid identity. The UTXO model provide that every node knows its balance by checking all the unused outputs inside the blockchain. As for POW, in our blockchain the level of difficulty for mining blocks is fixed, whereas in most decentralized ledgers that implement POW the difficulty is dynamic, so that the throughput of the blocks does not increase if the computational power raises. Usually this is done for security reasons: producing a lot of blocks could lead to a problem of propagation of the blocks in the network that would generate more orphan blocks or soft forks, and also problems such as double spending could be harder to prevent. In our blockchain, for development purposes only, the nonce (the difficulty) is fixed to 3: in this way a miner needs only to produce an hash of the block with at least three leading zeros. This can be easily achieved by increasing a counter-field inside the block header until the hash reaches the desired form, where the hash of the block corresponds to the hash of the block header. The entire mining pipieline will be explained in the node design section.

4.2 Sanity check design

A key ingredient for a blockchain to work properly is how every node validates transactions and blocks. Given that a faulty node could purposely propagate bad transactions or blocks, there should be some rules that good nodes can enforce on the network by rejecting bad transactions and blocks. In this way, if the majority of nodes respect this rule, the propagation of bad data would cease to spread around the network. For transactions, these are the rules:

- The transaction's syntax and data structure must be correct
- Neither lists of inputs or outputs are empty
- Each output value, as well as the total, must be within the allowed range of values
- The outputs being spent match outputs in the mempool or unspent outputs in a block in the main branch
- Reject if the sum of input values is less than sum of output values.

These are the rules for blocks:

- The block structure is valid if all the required fields are filled correctly
- The block head hash is less or equal than the target hash i.e. the number of leading zeros in the head hash greater or equal than the one in the target
- The block timestamp is greater than the MTP and up to 2 hours in the future
- The block weight is acceptable i.e. the number of transactions is less than a threshold
- The first transaction in the block is the coinbase transaction.

4.3 Recovery mechanism

The biggest problem in the blockchain design is what happens if some nodes (not the majority) of the network gets partitioned because of faulty communications, so that their version of the blockchain stays behind. Once they get back up, they'll have an outdated and invalid blockchain, so every transaction and block they receive it will be dropped. To recover from this situation, after a number of "bad blocks" is received, the node asks around for the current state of the blockchain. Even though we are in a decentralized network, thanks to the consensus algorithm we can just use the rule of the longest chain (in this case the blockchain with the most cumulative proof-of-work corresponds with the logenst chain, because we fix the difficulty for the mining procedure). So a node to recover the correct blockchain, just takes the longest chain and cross validates it with every node it knows, because we suppose at least more than half the network are not malicious nodes, we are able to recover the right blockchain.

4.4 Network design

The network design is simplified respect to real word blockchain for obvious reasons. First of all we designed a so called "DNS node" (that is also a miner node). When a new node joins the network it contacts the DNS node, so it can get a list of other nodes. Communication is done in a P2P way, where every message is transmitted to all other nodes.(fig. 1).

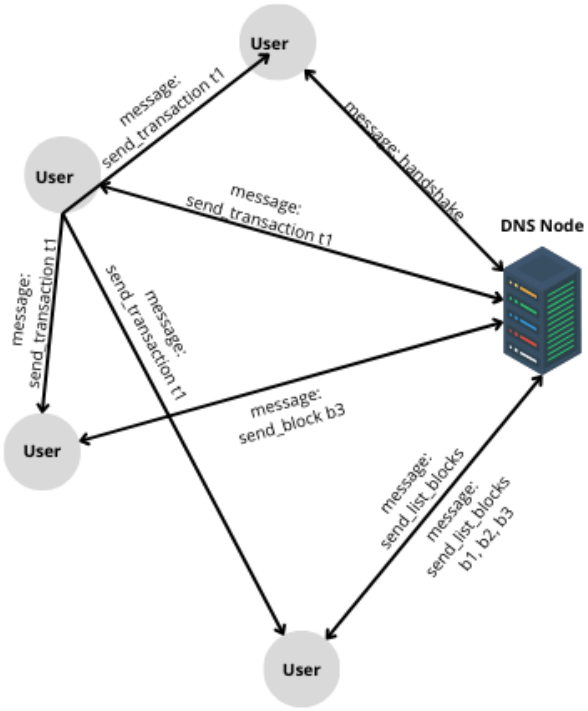

Figure 1: Network Design

4.5 Communication Protocol design

The communication protocol is composed of 5 messages:

- SEND TRANSACTION
- SEND BLOCK
- SEND LIST UP TO DATE BLOCKS
- SEND TRANSACTION MEMPOOL
- HANDSHAKE
- VERIFY LIST OF BLOCKS

The handshake message is the first message a user sends to the DNS node to acknowledge its existance to the network. The DNS Node responds with a list of existing nodes, that the new node can use to share or receive new transactions or blocks. Then we have the SEND LIST UP TO DATE BLOCKS this message is used to get the entire

blockchain of a node, it is really useful at the beginning when the node is new to the network. The same goes for the SEND TRANSACTION MEMPOOL, here a node share its pool of unverified transaction with the node. In case of a node lagging or receiving a false blockchain, it can recover the current state with the use of of VERIFY LIST OF BLOCK, where a node broadcasts the message SEND LIST UP TO DATE BLOCKS to all the nodes he knows and gets back a list of blockchains. The longest chain with most copies is the current one.

4.6 Node design

There are 2 types of nodes: a user node and a miner node. The main difference between these two types of nodes is that the latter mines blocks when the right amount of transactions gets reached. Each node has 8 operations it can execute:

- Register a domain
- Update the public offline key
- Update the public online key
- Revoke a specific domain
- Send money to a user
- Inspect the blockchain
- Inspect all the domains
- Inspect your balance

4.7 Structure / Domain Entities

The domain is composed as follows:

- **Block & BlockHeader**: data structure for rapresenting the block
- **BlockFactory**: used to generate the genesisBlock
- **MiningPipeline**: implements the mining procedure

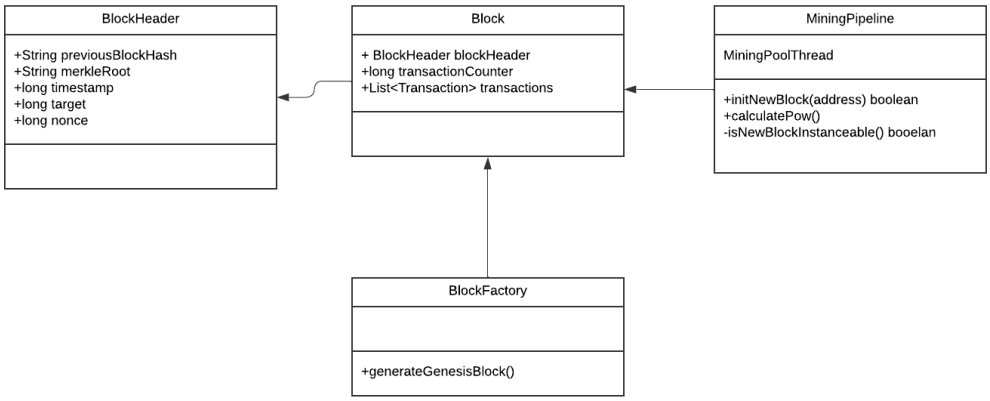

Figure 2: Zoom in block domain in UML

- **Node**: the abstract class that implements all the operations of a user/miner
- **ClientNode & MinerNode**
- **Client**: sends messages to other nodes
- **Server**: every node has a listening connection when it needs to receive data from other nodes
- **PKIProtocolActions**: implements all the main actions used for public key infrastracture
- **TransactionFactory**: used to generate transactions
- **TXInput & TXOutput**: indicate what transactions to consume for obtaining the amount of value needed

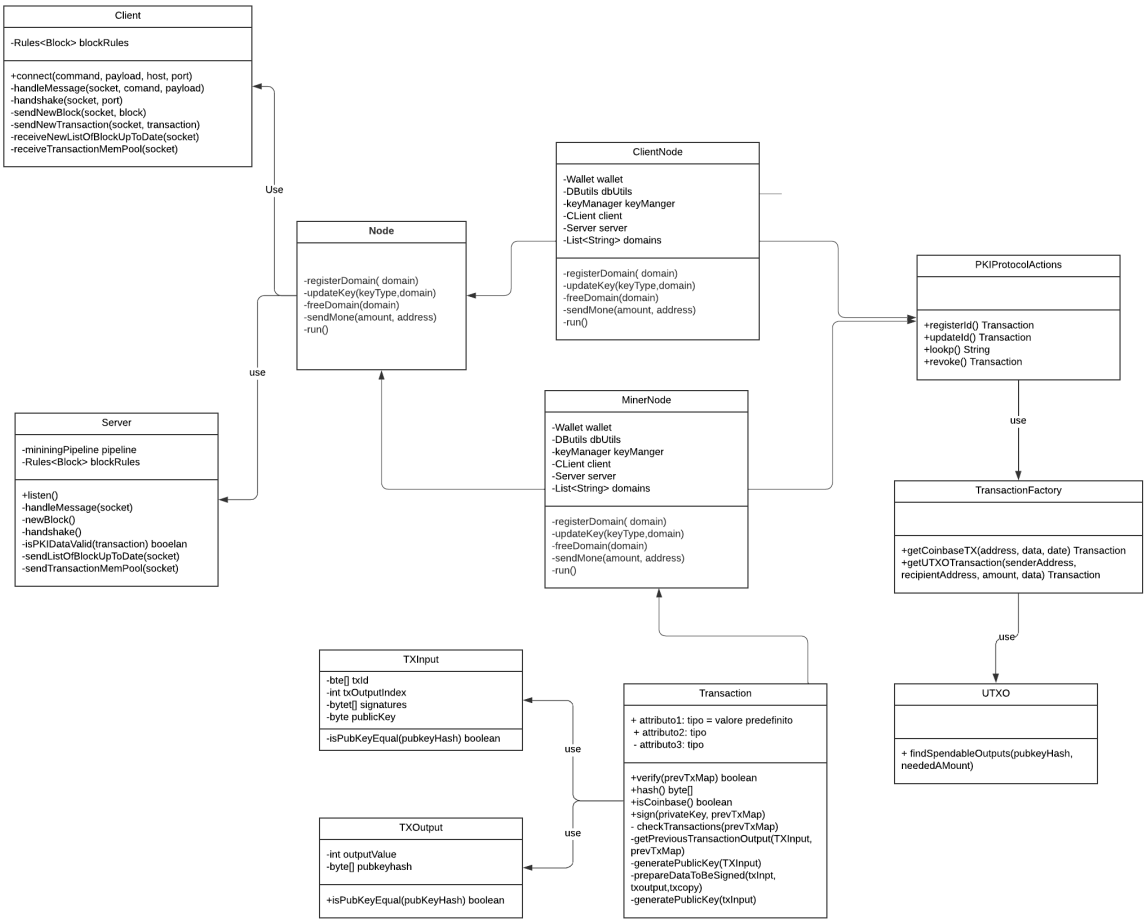

Figure 3: Domain in UML

5 Implementation Details

In the paper [1] there is no explicit instructions on how to implement a blockchain. We implemented a blockchain similar to bitcoin but with less features. Since we don't use Docker, we simulate the physical address of every node as a local port, rather than using containers with a private network. For saving the blockchain we use RockDB as a fast key value database.

6 Self-assessment / Validation

6.1 Testing the nodes functionality

Testing the correctness of the implemented operations is really simple, because most of the data we exchange eventually (if valid) gets saved in a block that gets appended in the blockchain; thus we can simply inspect the blockchain to look for errors.

6.2 Unit tests

As for unit tests, we mainly proved that all cryptographic operations work as intended and most operations depending on it, in particular:

- Signing operation
- Hashing
- Proof of work

7 Deployment Instructions

After you clone the repository, ensure that the wallet.dat file is present: it contains a list of registered accounts to choose from in order to simulate locally the existance of different entities. First of all we should start the miner

```
cd PKI/
./gradlew runMiner --console=plain
```

Then, we can start the clients (up to 18, beware they run locally so the number of threads needed increases)

```
./gradlew runClient --console=plain
```

Another word of caution: if the miner nodes stops for any reason you should restart all the other clients, and restart from a blank ledger.

8 Usage Examples

Once you start the client you have to choose which identity to use

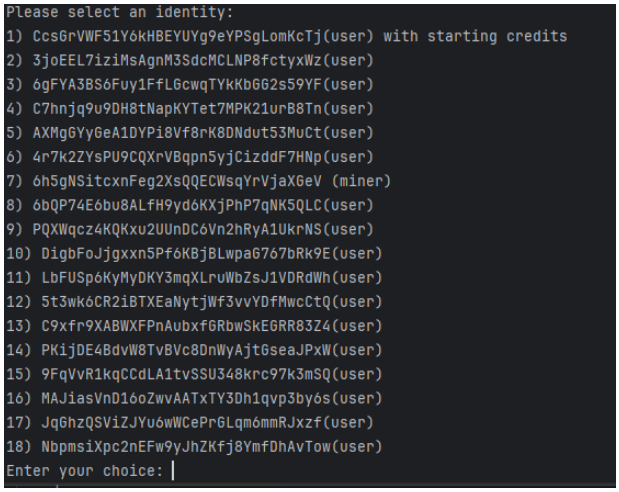

Figure 4: Identity selection

Once selected the identity you can choose whatever action to execute:

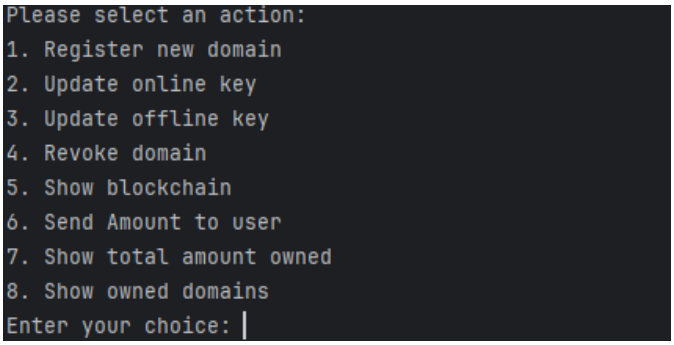

Figure 5: possible actions

Figure 6: Example of new domain

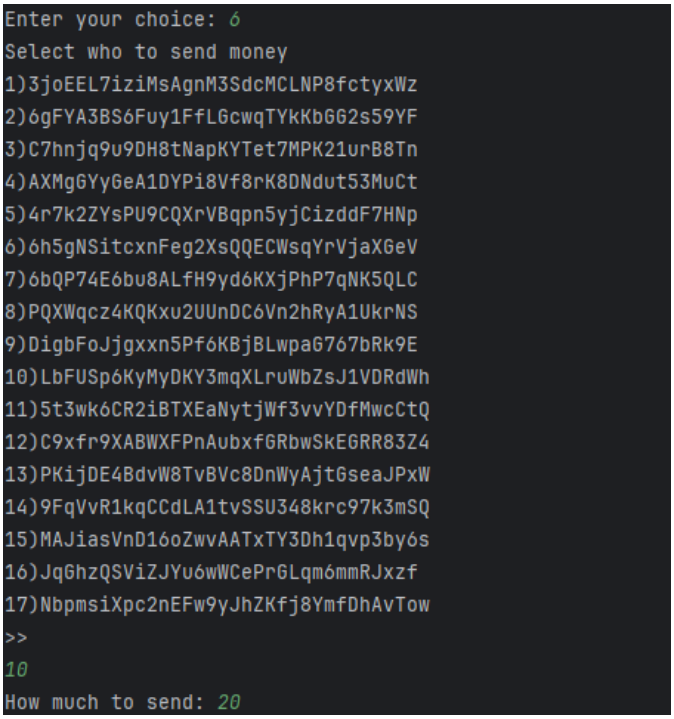

Figure 7: Example of sending money operation

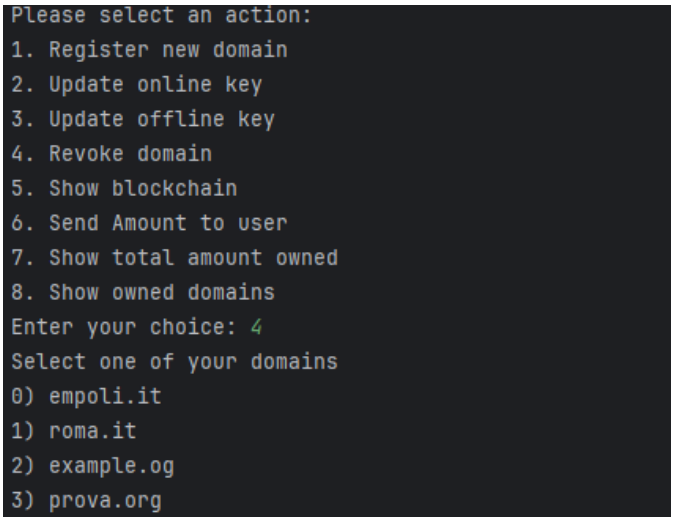

Figure 8: Example of revoking a specific domain

9 Conclusions

In conclusion, we managed to build a working homebrew blockchain and we achieved the successful implementation of the public key infrastructure. Our project is just a simple implementation of what is proven in the paper we took as inspiration [1] and the fundamentals of Bitcoin.

9.1 Future Works

In future works we would like to implement a more sophisticated communication protocol and simulate different devices using Docker containers. Another thing we could try to implement is another, less compute intesive, consensus algorithm like Proof of Stake.

9.2 What did we learn

During the development of the project we learned:

- **Blockchain**: the inner working of Bitcoin to understand how each component interacts
- **Cryptographic operations**: we used a lot of hashing and signing operations, for both consistency of the data and authenticity
- **Proof of Work**: how the process of minining is implemented and used inside a decentrilized ledger as a consensus algorithm
- **UTXO**: the UTXO model and how an account balance is represented as a series of input and output transactions.

References

[1] Conner Fromknecht, Dragos Velicanu, and Sophia Yakoubov. A decentralized public key infrastructure with identity retention. *IACR Cryptol. ePrint Arch.*, 2014:803, 2014.