
# Blockchain Based Approach for Preserving Car Maintenance History

Iva Najdenova

School of Computer and Communication Sciences

Decentralized and Distributed Systems lab

Master Project

March 2019

**Responsible**
Prof. Bryan Ford
EPFL / DEDIS

**Supervisor**
Linus Gasser
EPFL / DEDIS

**External Supervisor**
Alexandru Rusu
Swisscom

# Abstract

*Fighting frauds in the automotive industry is an ongoing challenge. Concerned by this problem are not only the owners and potential buyers of second-hand vehicles, but also entities like insurance companies, garages, car dealers, police etc. In our work, we present a solution for establishing trust between these parties, by keeping records of repairs and maintenance car checks in a decentralized ledger. For this proof of concept, we use the ByzCoin [15] blockchain protocol together with the Calypso [7] framework, which provides a secure way of storing and sharing confidential data over a blockchain with dynamic management of access policies and ownership of the vehicle's biography. The conducted evaluation of our implementation shows that the system works correctly also with larger networks, and up to 500 simultaneous car enrollments or report submissions.*

# Contents

# 1 Introduction

Upon buying a second-hand car, in most cases, consumers wonder whether that particular car has been stolen in the past or holds some hidden flaws. Frauds are very common when it comes to previously owned vehicles and some examples include tampering with the readings of odometers, hiding maintenance details as well as the history of accidents.

This problem in the automotive industry costs the consumers, car dealers, manufacturers, insurance and leasing companies approximately between 5.6 and 9.6 billion euros per year in Europe [17]. Thus, the question that persists is how to fight these criminal deceptions and establish trust among the above-mentioned entities.

Our motivation is to create a solution that allows buyers to be confident in their purchase and owners to increase the value of their car. We propose a proof of concept that includes a blockchain for keeping maintenance data of cars in form of a digital registry, where the records persist forever and are resistant to manipulation.

The idea is to add maintenance reports for every regular vehicle inspection, as well as for every unexpected malfunctioning or car crash. Some of the data used in these reports is generated by IoT devices that measure mileage, score of well driving and other parameters when placed on a vehicle.

This documentation is structured to offer a background information about the blockchain technology in section 2 and then introduce an overview of the project in section 3.

Furthermore, in section 4, we cover the implementation details of our work, including the management of access rights, creation of digital contracts for enforcing specific performance, handling the private data and also the interaction between the users and the blockchain.

Finally, in section 5 we test and evaluate the behaviour of our system in larger networks and with parallel transactions. We utilize visualized charts in order to draw better conclusions and then we finish our work with future possibilities for improvement.

In summary, this project shows how the blockchain technology can be applied in other spheres of the real world (particularly in the automotive industry), and not only in the well known financial area, with the purpose of solving trust and fraud issues.

# 2 Background

This section outlines the beginnings of blockchains applied in cryptocurrency systems like Bitcoin and Ethereum, and also describes the ByzCoin blockchain which was selected as most appropriate for this project.

## 2.1 Bitcoin Blockchain

The blockchain concept drew attention in 2008 when Satoshi Nakamoto introduced the Bitcoin [1] system for electronic transactions that rely on cryptographic primitives instead of trusting a third party. A brief description of the Bitcoin cryptocurrency system goes as follows:
Whenever Alice wants to transfer digital money ("coins") to Bob, she creates a transaction and broadcasts it to a peer-to-peer network. This transaction consists of a digital signature (using the private key of Alice) of the hash of the previous transaction related to these coins, together with the public key of the next owner - Bob [1].
Hence, Bob has access to the chain of ownership of the coins he has received, but still does not have a proof that Alice has not sent the same coins to another person. This double-spending problem is solved by requesting the nodes of the peer-to-peer network to agree on a single order of transactions, so that only the earliest transaction counts [1].

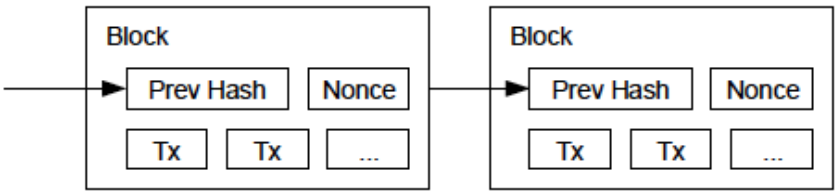


Figure 1: Blocks in Bitcoin [1]

Every participant of the network needs to first verify that the transaction initiated by Alice does not use already spent digital money, and then place it together with other transactions in a block that is supposed to be added on a chain of blocks (Figure 1). To save space, the transactions are hashed in a Merkle tree [26], and only the root of the tree is included in the header of the block. In order to append a new block, the node, also called miner, needs to solve a computing difficult task called proof-of-work (PoW) that involves scanning for a value (nonce) that produces a hash which begins with a determined number of zero bits [1]. As input to the hash function, in addition to the nonce, there is also the hash value of the previous block. This way, by hashing the transactions into a chain, it is impossible for attackers to change

a record without re-computing previous PoWs [1].

It can happen that two or more participants find a proof-of-work at nearly the same time, and this is going to create forks in the blockchain. To solve this issue, only the longest branch of blocks and the transactions taking part of it is valid. Hence, the historical order of transactions is unique and the same for everyone.

A transaction in the Bitcoin blockchain is considered to be confirmed with high probability only after six additional blocks have been appended behind it. The block mining rate is 10 minutes on average. This introduces high latency in the system and hinders real-time transactions (Imagine waiting one hour for a coffee you have bought with your digital coins).
Some other inconveniences also exist like waste of resources caused by the unsuccessful miners when they are trying to append a new block. Furthermore, even though the Bitcoin blockchain documentation [1] ensures correct processing of transactions if the majority of computing power is not owned by malicious miners, there are still some vulnerabilities like strategic mining attacks [19, 14].

## 2.2 Ethereum Blockchain

Since the Bitcoin cryptocurrency system created a revolution in the world of decentralized systems, many other blockchain protocols and applications started appearing, either to improve on Bitcoin or to find usage in other fields (e.g. health care [2]).

One of the nowadays best known blockchain protocols is Ethereum [16], due to a universality principle that allows users to create their own rules for state updates ("contracts") using internal scripting language and then apply these smart contracts in a variety of blockchain based applications (financial, semi-financial and not financial at all, like online voting and decentralized governance) [16].
The straightforward way of creating digital contracts that autonomously enforce the rules of interaction and are precompiled in the nodes that maintain the blockchain [16], is one of the most significant benefits one can get for choosing Ethereum over the Bitcoin blockchain protocol.

Ethereum can be interpreted as a transaction-based state machine where every new valid transaction leads to a state transition [16]. Similarly, like in Bitcoin [1], for appending a new block of transactions, a miner needs to perform a computationally difficult task, with the underlying algorithm Ethash, in order to find a proof-of-work before other competitors. An advantage of Ethash is that it is ASIC-resistant, meaning that finding a nonce requires a

lot of memory and it is not possible to use the same memory for discovering multiple nonces in parallel. Accordingly, this allows better decentralization by making the distribution model as open as possible and preventing malicious miners to skew the distribution in their favour [16].

Overall, Ethereum guarantees that every two individuals can be confident that the outcome of their interaction will be in coordination with the specified digital contract.

### 2.3 ByzCoin Blockcain

The proof-of-work technique utilized in the Bitcoin blockchain protocol [1], solves the consensus problem - several processes deciding on one value from a set of proposed values (in this particular case: which update of the state of the distributed ledger to follow). However, it only provides probabilistic consistency guarantees, meaning that clients cannot immediately be certain that a submitted transaction is committed even though it has been added on the blockchain [15]. As mentioned in section 2.1, it takes at least one hour for a transaction to be considered confirmed with high probability.

The blockchain introduced by the EPFL DEDIS (Decentralized and Distributed Systems) Laboratory solves this transaction commit latency problem, by replacing Nakamoto's consensus [1] with a Practical Byzantine Fault Tolerance [24] based protocol called ByzCoinX [22]. An improvement on top of PBFT, that reduces the costs of the multiple rounds of this protocol, is the inclusion of CoSi [18] - a collective signing protocol that aggregates thousands of signatures [15]. ByzCoinX uses collective signing to achieve consensus with absolute finality property (transactions are irreversibly committed within seconds unlike with probabilistic finality), and at the same time preserves the open membership characteristic of a public ledger like Bitcoin [15].
The servers that run the ByzCoinX protocol and offer decentralized services to users, are called collective authority nodes or later on referred to as conodes.

In this protocol, there is a leader which is responsible for contacting every other node in the system in order to collect transactions submitted by end users. For the communication, as well as for the collective signing, ByzCoinX uses a tree structure with the leader as root, which significantly reduces the transaction confirmation latency in the system [15].
Once the leader receives transactions from the followers, a block containing the transactions is being created and sent back to the followers for validation. If there are remaining transactions, they will be considered upon creation of the next block [4].

In case the leader starts behaving maliciously or simply does not perform as expected, the followers initiate a **view change** (change of the leader) [4].

In a similar fashion like in Ethereum [16], the ByzCoin blockchain protocol works with smart contracts precompiled into the conode binary, that define how a certain instruction updates the global state. The leader computes this state change, but every conode verifies whether it is correct. The global state consists of instances, each one of which is tied to a contract and holds data (Figure 2). A Merkle tree [26] root of the global state is stored in every header of a block.

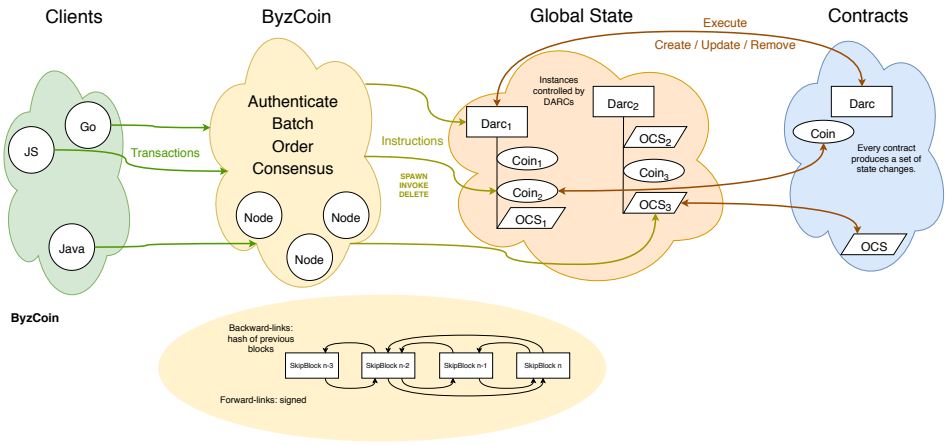


Figure 2: ByzCoin (Simplified version of [5])

Instead of using a standard chain of blocks, where only neighbors are connected, the DEDIS team presents a data structure that combines blockchains and skip-lists [13] and names it a **skipchain** [8].

Skipchains enable both backward and forward timeline traversal with single or multi-hop links. The backward links are cryptographic hashes of past blocks and the forward links are cryptographic signatures of future blocks that are added retroactively when the target block appears [8]. These forward links allow future block validation, which is especially useful in scenarios such as software updates, where an outdated client gets an update from a not necessarily trusted peer [8].
The long-distance links allow traversal in a lower number of steps (logarithmic cost) and provide shorter proofs of existence. ByzCoin is, so far, the only blockchain technology that uses this concept in order to prove that a given instance exists without having to know the whole blockchain. Skipchains also enable clients to verify that transactions are on the blockchain without

the need of being connected to the Internet [8].

Another clear benefit that ByzCoin [15] blockchain provides is management of the access policies in the system in a dynamic and fully decentralized manner. As a substitute for a password or public key for authentication, a DARC (Distrubuted Access Right Control) structure is being used [12]. It allows evolution of the description about who can or cannot access a certain resource, by maintaining a set of changeable rules. How these DARC structures are implemented is explained in details in section 4.1.

Taking into consideration that most of the existing blockchain applications which contain sensitive and private data use either semi centralized approaches (secret data is not stored on the chain) [20, 7] or ignore the privacy problem [3, 7], the DEDIS Laboratory introduces a framework called Calypso [7]. Calypso provides a secure way of sharing confidential data over a blockchain and also prevents a single point of compromise or failure in the system by keeping the private data as on-chain secrets (using threshold cryptography) [7].

The primary goal of Calypso is to allow only authorized clients to decrypt the on-chain secrets, while maintaining at the same time a tamper-resistant log of every access transaction [7].

The overview of the Calypso framework architecture is shown in Figure 3. It involves a writer, a reader and two collective authorities (groups of signers) [18].
The first one is called **access control cothority** and is responsible for verifying and logging write and read transactions, as well as enforcing access control policies. It consists of the same conodes that run the ByzCoin blockchain protocol, but these servers also have the Calypso read and write contracts compiled in their binaries.
The second one is named a **secret-management cothority** and is in charge of handling and delivering secrets.

The process of storing and reading on-chain secrets goes as follows:

0. Initially, an administrator generates collective private-public key pair for the secret-management cothority, by running a Distributed Key Generation algorithm.

1. The writer creates a symmetric key to encrypt his/her secret and then encrypts this symmetric key with the collective public key of the secret-management cothority [7].

2. The encrypted secret and its access policy are stored on the blockchain and a reader is able to download the secret and request to read it. The read request is verified by the secret-management cothority and then logged on the blockchain.

3. The reader uses the logged read request to ask the secret-management cothority to re-encrypt the initial symmetric key with his/her public key.

4. The reader can decrypt the symmetric key and therefore decode the secret.

Providing a secure sharing of sensitive data with dynamic management of access policies and ownership, is what makes Calypso and the ByzCoin blockchain a perfect match for our project.

Car owners are able to grant access to their car service history to potential buyers in order to increase the value (price) of the car, with the possibility to remove the access if the buyer is no longer interested. Additionally, every read and write transaction is logged and the history of the system cannot be compromised if less than $\frac{1}{3}$ of malicious nodes are present.

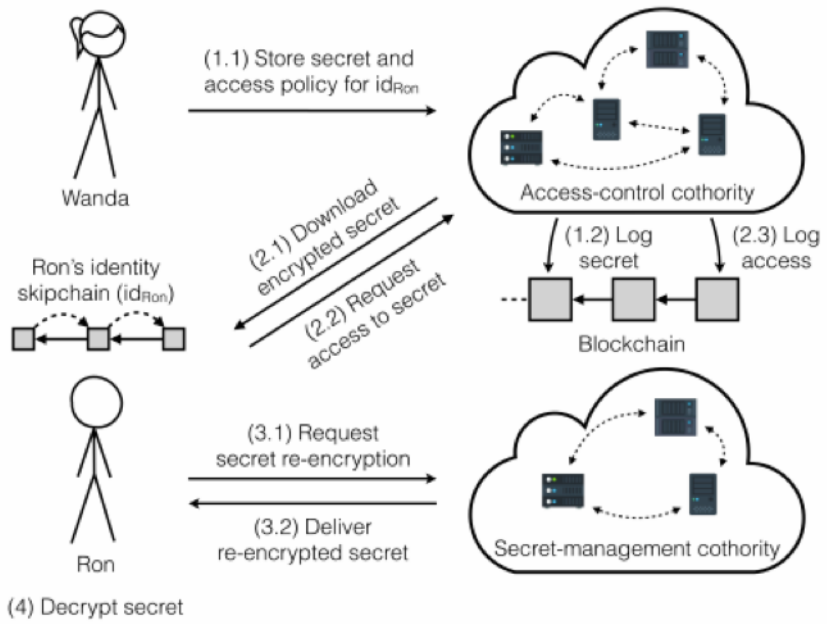


Figure 3: Calypso Overview [7]

# 3 Project Overview

As stated in the introduction, the goal of our work is to implement a blockchain solution that can be used to fight frauds in the automotive world. To be more specific, we apply the ByzCoin [4] blockchain technology and build a service on top of it.

For a correct implementation, the system requires at least 5 machines, with the assumption that at most one of them might be faulty or byzantine. This is deduced from the statement that ByzCoin tolerates $f$ faulty members among $3f + 2$ nodes in the system [15]. These conodes are envisioned to be distributed between the distrustful parties (car manufacturers and dealers, insurance companies, police...).

Each machine maintains a local copy of the blockchain and users are able to interact with the nodes by using a desktop application (Figure 4). They can either create and send a new transaction, or get a proof of existence for the data stored on the blockchain.

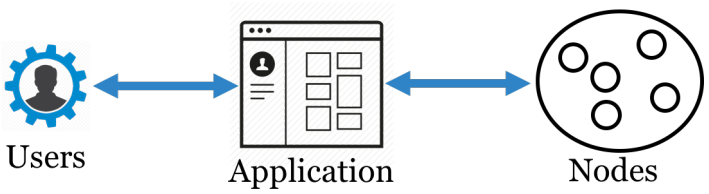


Figure 4: System Description

In what follows, we describe the use case of our project.

## 3.1 Use Case

Depending on the access control in the system, blockchains can be public or private. Private blockchains are suitable only when a set of known players is allowed to participate. What ByzCoin and Calypso bring into the picture is the possibility to open the system to unknown participants.

Current limitation, which is not inherent to the system, is that an administrator, at the beginning, defines which conode machines are going to be part of the system and then sets the blockchain up. The administrator is also in charge of enrolling new cars and members, each time a new user decides to join the service. This accounts for a point of making profit, if registration charges are introduced.
There are several roles that the participants can take on (Figure 5):

- vehicle owner
- a person allowed to add maintenance reports (further on referred to as a garage mechanic)
- potential buyer

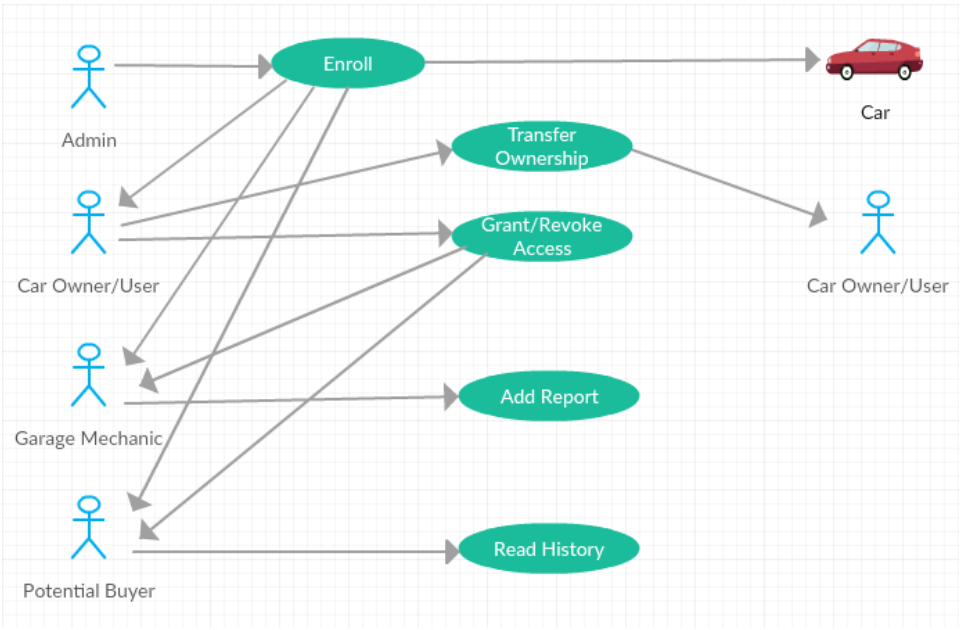


Figure 5: Use Case Diagram

The car owner is responsible for granting and revoking read access to potential buyers and write access to garage mechanics. Additionally, the ownership of the car history can easily be transferred to another user after a successful car sell without involving the administrator.

Furthermore, garage mechanics are responsible for data generation during every regular car inspection or unforeseen repair, and are also required to use the information provided by the IoT device connected to the car.

Finally, the entire maintenance history of the vehicles can be requested and read by the allowed potential buyers.

Having envisioned and described the system model, we are able to proceed with specifying the implementation details.

# 4 Implementation

The starting point of our implementation is the Cothority Template [11], which serves as an example on how to create a ByzCoin [15] blockchain service.

The main elements of the ByzCoin implementation are: **instructions** sent by the clients, **contracts** that define how the instructions are interpreted, a global state with **instances** which are tied to a contract and hold data, and **DARC structures** (explained bellow) for access control (Figure 6).

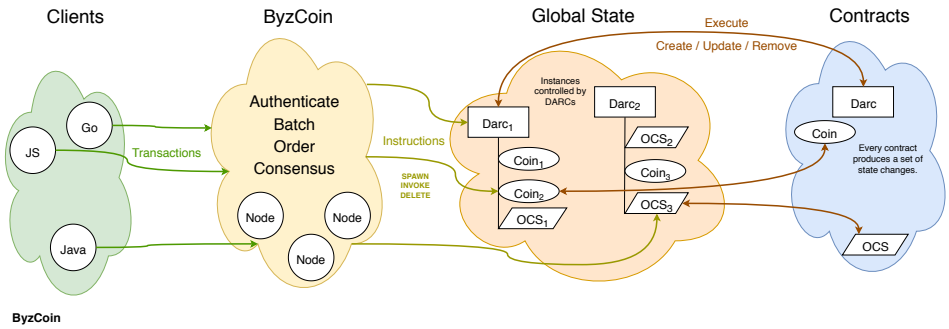


Figure 6: ByzCoin (Simplified version of [5])

Every user instruction which is sent to the ByzCoin service can be one of: **spawn** for creation, **invoke** for update or **delete** for removal, and needs to have an existing instance as a target (Figure 6). These instances maintain information about their unique **ID**, **version** which is the number updates, the **contract ID** tied to that particular instance, **data** to be interpreted by the contract and a **DARC ID** for access control [9].

A detailed description of the ByzCoin implementation can be found on the DEDIS cohority git repository [4].

## 4.1 Access Control with DARCs

The management of access rights is handled by DARC structures (Distributed Access Right Controls) [12], which maintain a set of rules. Every rule consists of an action name and an expression that contains identities allowed to execute that action.

DARCs can be updated by the identities specified in the **evolve** rule. There exists also a notion of delegating the permissions to another DARC, by specifying the DARC ID in the rule expression instead of an identity. This delegation plays an important part in the smooth transfer of car history ownership, that does not require an intervention by the administrator. On Figure 7,

we can observe the schema of DARCs required for our project.

Due to the security guarantees of our system, only the administrator has the possibility to create new DARCs with the **spawn:darc** command. Moreover, each participant (user) has a dedicated DARC, that will prevent losing his/her access rights upon losing credentials (private key), by including the administrator identity in the **evolve** rule.

Furthermore, for every enrolled car, there needs to exist a separate DARC for storing owner, writers (garage mechanics) and readers (potential buyers), as well as a car DARC with rules for: creating a car instance, adding reports, adding Calypso secrets and reading Calypso secrets.

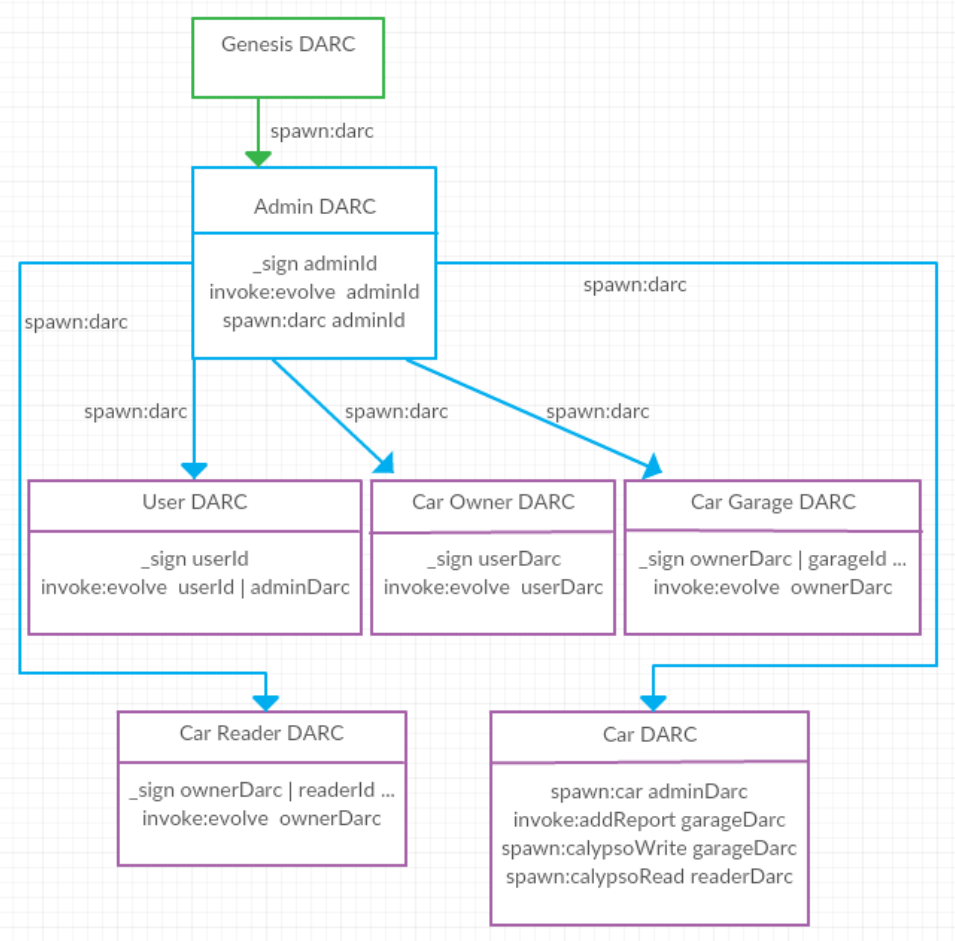


Figure 7: DARCs Schema

Now that the DARC schema is determined, we can proceed to creating the digital contracts that define what happens in the global state once some particular command is invoked.

## 4.2 ByzCoin Contracts

The term smart contract is being used for digitally enforcing specific performance. ByzCoin contracts resemble the Ethereum smart contracts [16] with the difference that they are precompiled and every node needs to have the same version in the interest of achieving consensus [9].

Digital contracts [9] regarding the ByzCoin configuration, the access control and the Calypso functionality are predefined and take part of the cothority project [10]. What is missing for our service is an additional contract related to the car instances, which had to be first designed and then registered with ByzCoin.

Within this car contract, we define methods for the **spawn** and the **invoke** type of instructions which create or update car instances accordingly (Section 4). A method for the **delete** instruction is intentionally omitted in the car contract as we wouldn't like any vehicle's maintenance history to be erased.

Before creating a new car instance in the global state, we firstly check whether the data corresponds to a car object with a string of vehicle identification number (VIN) and an empty list of reports (Table 1). If the verification is correct, a new state change is being made. The command for this instruction is named **spawn:car**.

When updating a car instance, a report is being added to the list of reports about that particular car. Every report consists of a string representing the ID of the garage mechanic that submitted the report, the date of the maintenance check and a calypso write instance that keeps secret data like for example: mileage, repairs, warranty, etc (Table 1). The command for this update instruction is named **invoke:addReport**.

| SecretData | Report | Car |
|---|---|---|
| string ECOScore | string Date | string VIN |
| string Mileage | string GarageID | [ ]Report Reports |
| boolean Warranty | [ ]byte WriteInstanceID | |
| string CheckNote | | |

Table 1: Data Structures

With this being specified, the car contract is ready to be registered.

### 4.3 Calypso Interaction

Before we dive into the usage of Calypso in our system, let us first take a look at how Calypso is actually implemented [6].
As presented in the section 2.3, Calypso requires two collective authorities:

- Access Control Cothority
- Secret Management Cothority

In our implementation, we use the same nodes to constitute both access control and secret management cothorities.

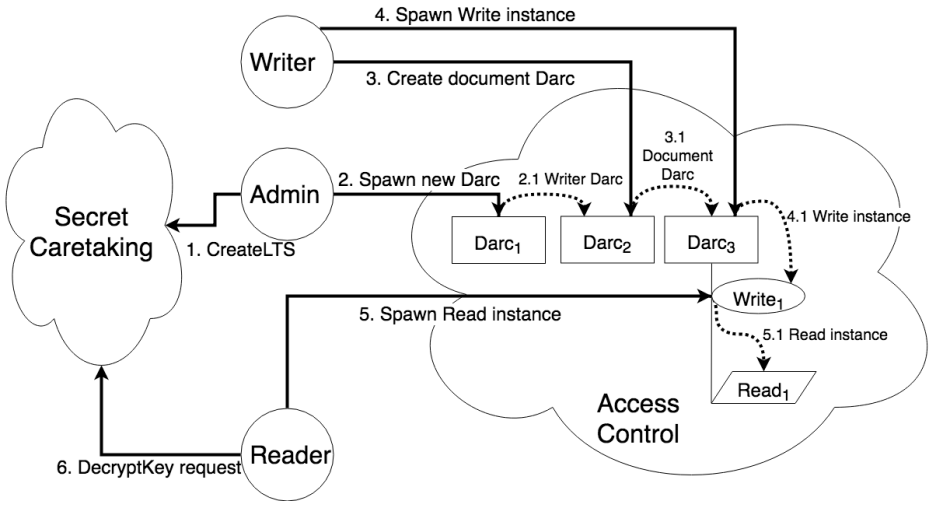


Figure 8: Calypso [6]

As mentioned before, at the initialization of the blockchain service, the administrator needs to generate a distributed key pair for the secret-management cothority (Figure 8 Step 1). Every writer in the system (garage mechanic from Figure 5) encrypts the symmetric key of his encoded secret with this collective public key.

Furthermore, the access control cothority runs the ByzCoin protocol and also maintains calypsoRead and calypsoWrite contracts. Thus, readers (potential buyers from Figure 5) and writers (garage mechanics) can use it for spawning read and write instances accordingly, as well as DARCs for access control (Figure 8 Steps 2, 3, 4 and 5).

Finally, having the Read and Write Instance IDs, the reader (potential buyer)

is able to request the secret-management cothority to re-encrypt the symmetric key used to encrypt the secret by the garage mechanic, with the public key of the reader in order to be able to decode the secret data (Figure 8 Step 6).

With this being explained, we can continue with the inclusion of Calypso into our project.

The car ByzCoin contract enables creation of car instances. Once they exist on the blockchain, the authorized garage mechanics are able to add reports about them.
Adding such maintenance reports consists of two steps, i.e. two different instructions (Figure 9):

1. Spawning a Calypso Write Instance that holds the private data (encrypted using symmetric encryption)

2. Updating the Car Instance by appending a new report to the list

In order to read the car "biography", a potential buyer needs to go through the blockchain and find the block where the car instance is stored. This process is much faster when using the DEDIS skipchains [8], thanks to the long-distance links. The following steps are required for the potential buyer when reading a maintenance history:

1. Obtaining the car data (VIN and list of reports)

2. Spawning a new Calypso Read Instance, by using the Write Instance ID from the report

3. Requesting the re-encryption key from the Calypso Secret Management Cothority, by providing the Read and Write Instances

4. Receiving the re-encryption key and decoding it by using his/her private key

5. Decrypting the secret

For each report, steps from 2 to 5 need to be repeated.

Thanks to Calypso, it is cryptographically hard to learn any information about the maintenance histories of the cars on the blockchain, unless having the permission to do so.

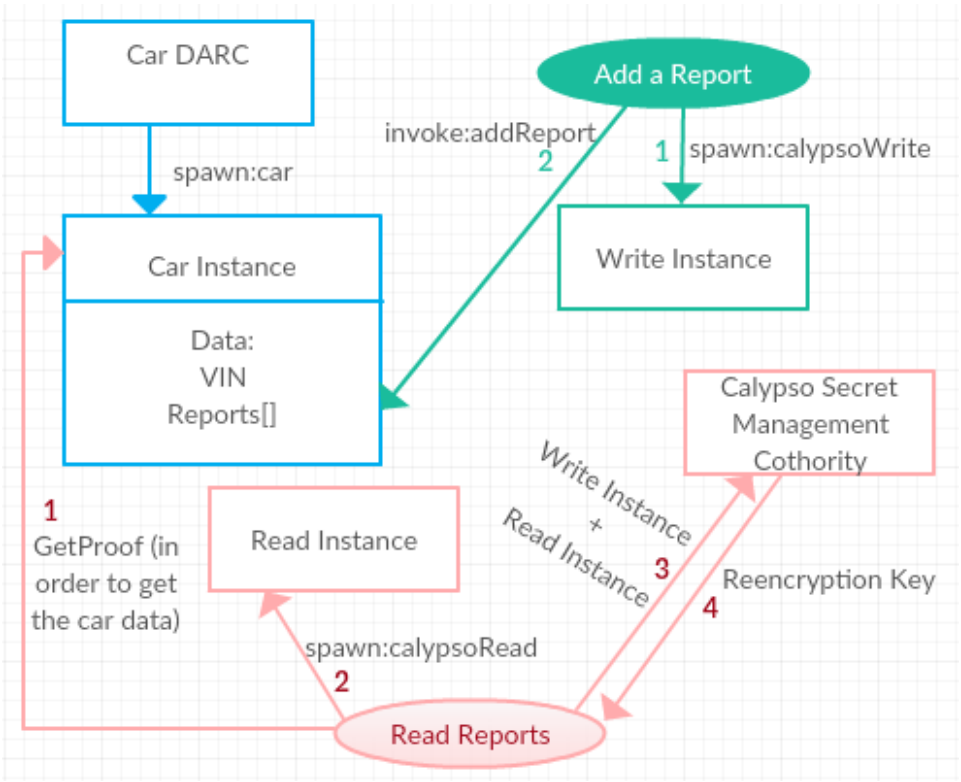


Figure 9: Add Reports and Read History

### 4.4 Client Application

Having the car contract defined, together with the DARC structure for the system, we were able to create unit and integration tests in order to make sure everything works as expected (including the interaction with Calypso). Once these tests passed and proved the correct functionality of the system, we were confident about moving forward with developing a user friendly interface for communication between the end users and the distributed ledger.

In order to implement a desktop application, we decided to use the JavaFX software platform. To this end, we needed to use protocol buffers (mechanism for serializing structured data[25]) for a conversion from "Go" to "Java", and we also repeated our unit and integration tests, but this time in the Java programming language.

We designed the desktop application in such a way that the necessary configuration data, as well as the private credentials are stored locally in JSON format.

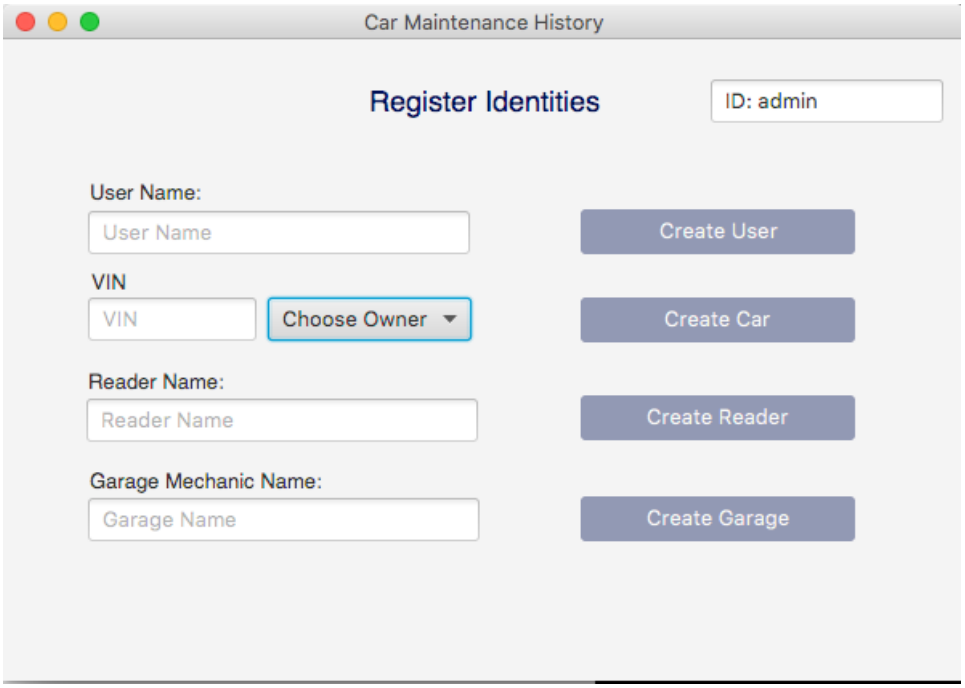


Figure 10: Admin Screen

On the home screen, for demo purposes, it is possible to choose the identity from all enrolled participant and the administrator. The administrator is able to further register new cars, as well as new members (Figure 10). As a reminder from chapter 4, it means adding all necessary DARCs for access control in the global state. The new members can either be: users with potential to become car owners, readers with the possibility to request car "biographies" or garage mechanics intended to add maintenance reports.

A car owner can use the desktop application to grant and remove access rights to potential buyers and garage mechanics for every vehicle he/she possesses (Figure 11). It is also possible, on the same screen, to transfer the ownership of the selected car. Each one of these options updates the responsible DARC with the **invoke:evolve** command.

Once a garage mechanic has been granted a write permission, he/she is able to generate review about the current state of the checked vehicle and place it in a blockchain transaction (Figure 12).
Currently, a report consists of data about the mileage, warranty, score of how well the vehicle has been driven and a check-up note, but it is always possible to add additional parameters like for example the number of accidents.

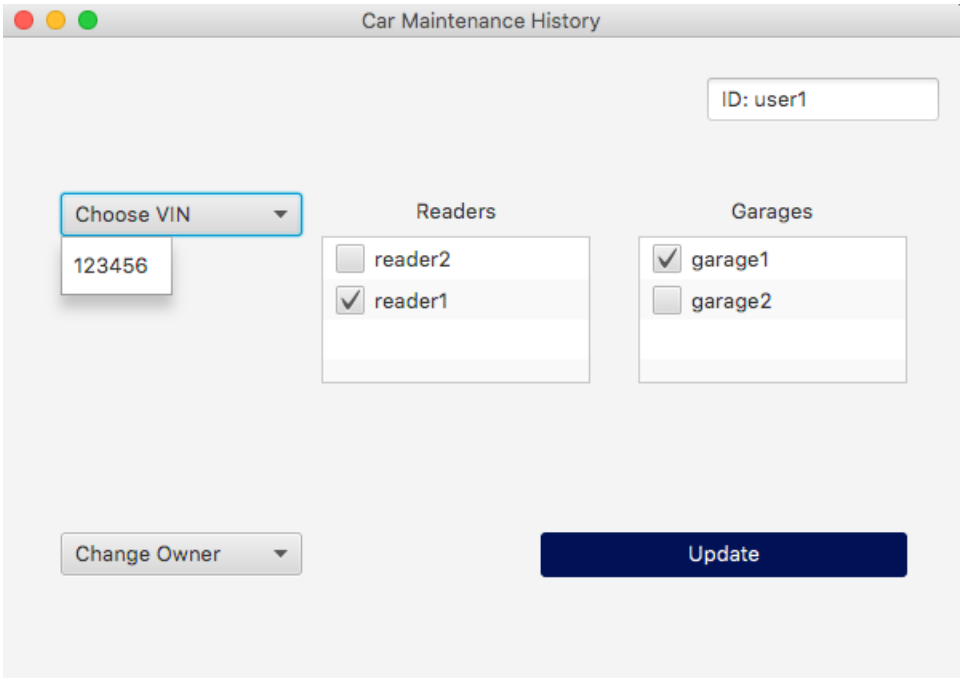


Figure 11: Car Owner Screen

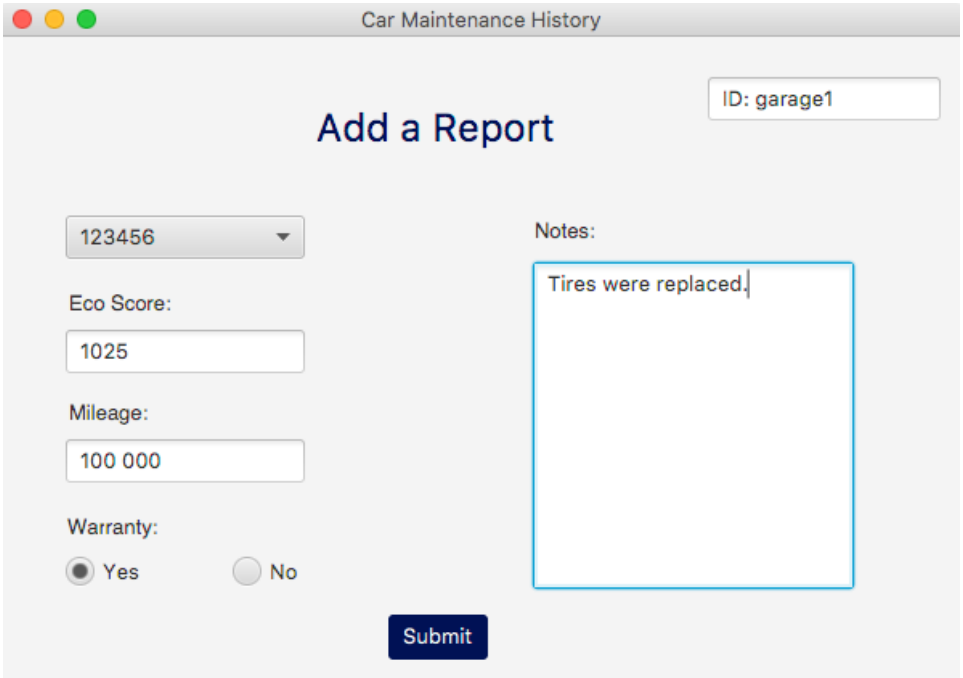


Figure 12: Garage Mechanic Screen

When a potential buyer enters the application, it is possible to request the maintenance history of the chosen car (Figure 13). If he/she is not allowed to access it, the application shows an error.

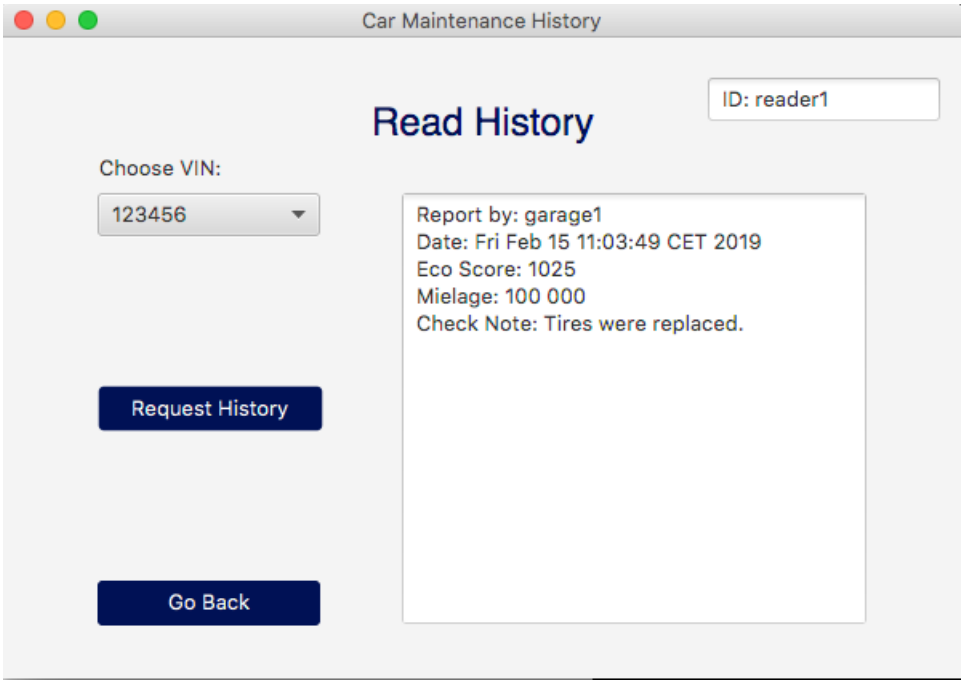


Figure 13: Potential Buyer Screen

## 4.5 Integration with the ByzCoin Skipchain Explorer

Many blockchain implementations include a tool that enables users know detailed information about which transactions are taking part of a particular block.

For our project, we are connecting the car ByzCoin blockchain to a SkipChain Explorer [28], that has been developed as a student project in the DEDIS Laboratory at EPFL.

On Figure 14 we can see the current interface of the SkipChain Explorer, which contains the block number, transactions with instructions (**addReport** in this case), status (accepted or rejected), as well as signatures.

It is also possible to visualize the skipchain as a graph made of blocks, and load information only about the blocks we are interested in.

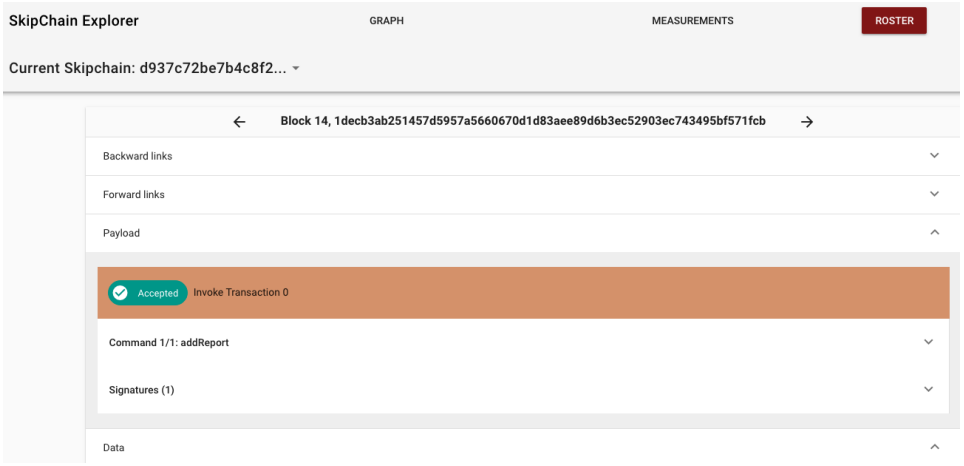


Figure 14: SkipChain Explorer

After being confident about the correctness of our implementation, we were able to proceed with evaluating our work in a more realistic setup (larger networks).

# 5 Evaluation

Having developed a fully functional ByzCoin Blockchain which keeps immutable data about vehicles, leads us to testing our solution in a larger network with different number of conodes and also distinct number of concurrent transactions.

As hardware for our simulations we used the IC Cluster (`https://iccluster.epfl.ch`) at EPFL. Depending on the configuration, some nodes might need to be run on a same physical server. Controlling the bandwidth and delay of the network, not only between the servers, but also between every virtual node, is done thanks to the Mininet [21] platform. Each server has 24 cores, 2.5 GHz processor and 256GB of RAM and can run around 300 cothority nodes simultaneously [21].

When running a simulation, a configuration file is needed. It should contain specific parameters like the name of the simulation, the number of: servers, hosts, transactions etc, as well as the before mentioned delay and bandwidth between nodes.

For our experiments, we needed to implement the **onet.Simulation** [23] interface, and specify in the **Run()** method what scenario should happen. In the beginning, it is mandatory to create the genesis block, and set the blockchain up by specifying the time interval for creating blocks and generating the collective distributed key for Calypso [7]. Part of the preparation is also enrolling the administrator and a user that will take the roles of owner, potential buyer and garage mechanic for simplicity.

Next step is deciding which measurements we are interested in. For each one of these measurements, we store in a .csv file the wall time (number of seconds it takes in real life, with the network communication included) and also the system cost calculated in seconds (time during which the system nodes utilize their CPUs).

First, we want to determine the time it takes to concurrently enroll a certain number of cars (defined in the configuration file). Registering vehicles consists of two different transactions: one for creating car DARCs that manage the access policies and another for creating car Instances in the global state.

Furthermore, we measure the time needed to add reports in parallel for each one of the registered cars. This is also done with two transactions by producing first Calypso Write Instances and then updating the car instances to contain the report with the secret data.

Finally, we record how long it takes to read the maintenance history simultaneously for every enrolled car. Here, we distinguish between the time needed for inserting a Calypso Read Instance into the global state (one transaction), and the time required to obtain the re-encryption key from the secret management cothority in order to decode the secret (no transactions are created for this step).

We have combined and plotted these measurements, so that it is possible to observe how the time differs.

## 5.1 Changing the Number of Car Enrollments

In this section we describe a simulation where the number of hosts remains constant - 5, but we modify the number of concurrently registered vehicles (100, 200, 300, 400 and 500). The bandwidth we have configured is 100 MBps (both sending and receiving), whereas the delay between every two hosts is 100 ms.

After conducting the planed experiments, we visualize the wall time (Figures 15 and 16) and also the system cost (Figures 17 and 18) measured for a single: car enrollment, report addition and reading of a report. As a reminder, two transactions are needed for each car enrollment and each report added, whereas only one transaction is needed for reading the secret data.

On Figures 15 and 16, it is perceivable that the wall time calculated per car enrollment or report added/read is greater when there is a lower number of concurrent transactions. This result was expected due to the fact that the block creation time is equal for every case. We can draw a conclusion that the time for executing one transaction stays approximately the same in the scenarios with more than 300 parallel car enrollments, because the blocks are filled with the maximum number of transactions that can be fitted inside them. It is also important to point out that the system crashed upon stressing the network with 1000 simultaneous enrollments.

Observing the re-encryption time needed for reading one secret in Figure 16, we can say that it stays constant in the scenarios with different number of parallel requests addressed to the secret management collective authority. This outcome was expected, because the number of hosts remained unchanged and no transactions were involved in this process.

Regarding the system cost (time during which the nodes execute instructions), we can see on Figures 17 and 18 that it remains constant or slightly increases with greater number of concurrent transactions. This behaviour was presumed, as we don't consider the waiting time for block creation.

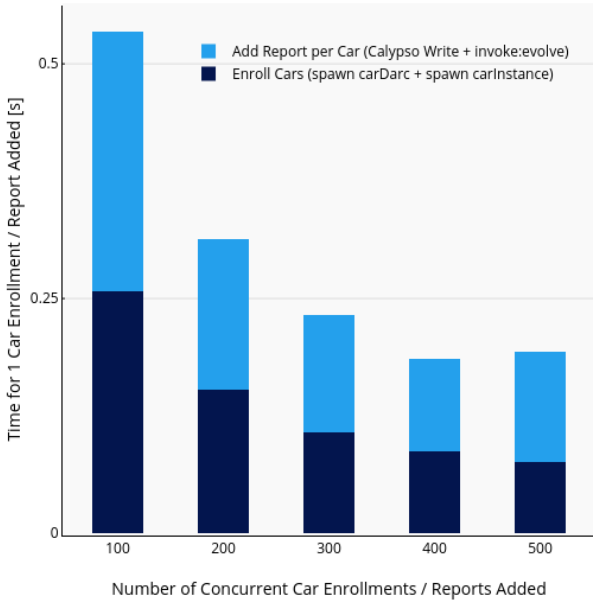


Figure 15: Wall Time for a Car Enrollment/Report Added with 5 Hosts

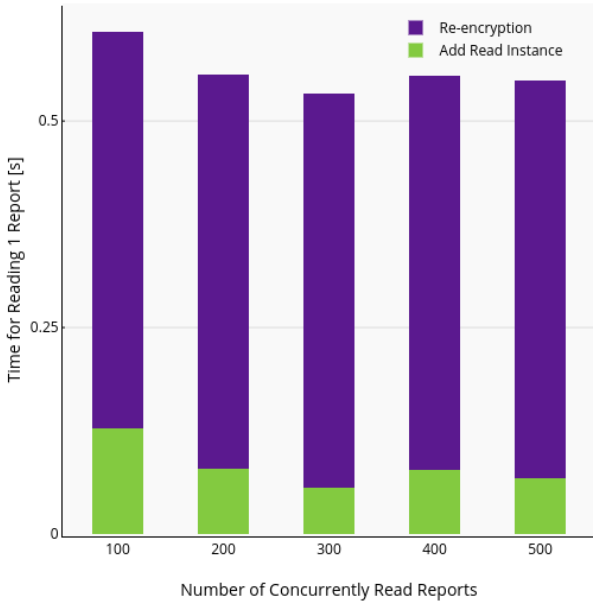


Figure 16: Wall Time for Reading a Report with 5 Hosts

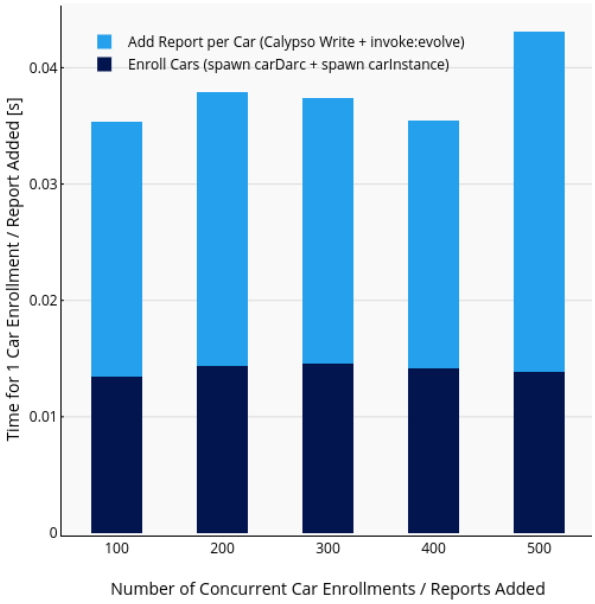


Figure 17: System Cost for a Car Enrollment/Report Added with 5 Hosts

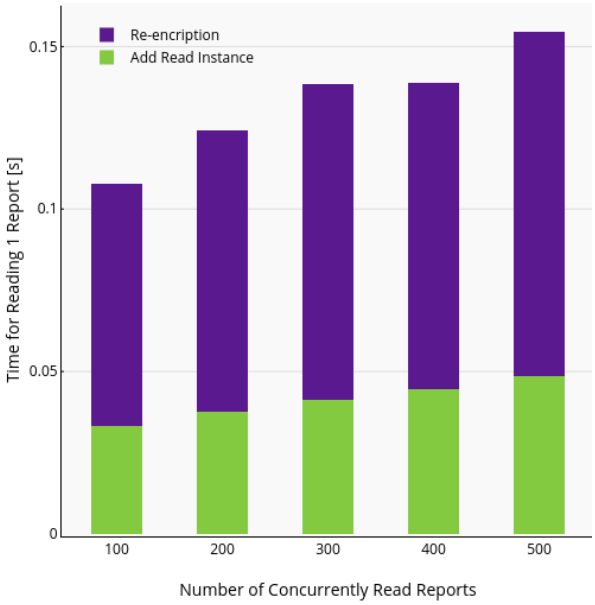


Figure 18: System Cost for Reading a Report with 5 Hosts

## 5.2 Changing the Number of Nodes

In this section we take into consideration a simulation where we modify the number of nodes that maintain the blockchain (5, 10, 20 and 40), whereas the number of concurrent car enrollments remains constant - 100. The bandwidth we have configured is 100 MBps, whereas the delay between nodes is 30 ms.

Having performed the experiments, similarly as in the previous section, we were able to visualize the wall time (Figures 19 and 20) and also the system cost (Figures 21 and 22) measured for a single: car enrollment, report addition or reading of a report, but now in a setup with different number of hosts.

From all the figures in this section (Figures 19, 20, 21 and 22), we can deduce that enrolling cars, adding reports and read instances in the global state, does not depend immensely on the number of hosts when there is a fixed number of simultaneous transactions. This is logically expected when every host is executing the same type of instructions.

What makes a big difference (for both wall time and system cost) is requesting the re-encryption key from the secret management collective authority (Figures 20 and 22).

The reason behind this performance, is that in our implementation, the same nodes are utilized to form both the ByzCoin collective authority (access control) and the Calypso secret management cothority. Thus, the re-encryption process takes longer with more nodes, as the broadcasting protocol involves many members.

An improvement would be to use an optimal, fixed number of hosts that constitute the secret management collective authority, uncorrelated to the ones forming the access control cothority.

By conducting these simulations, we gained a better perspective about how our solution works in larger networks and with different number of concurrent transactions. We have also discovered limitations of the system, like not supporting 1000 simultaneous car enrollments.

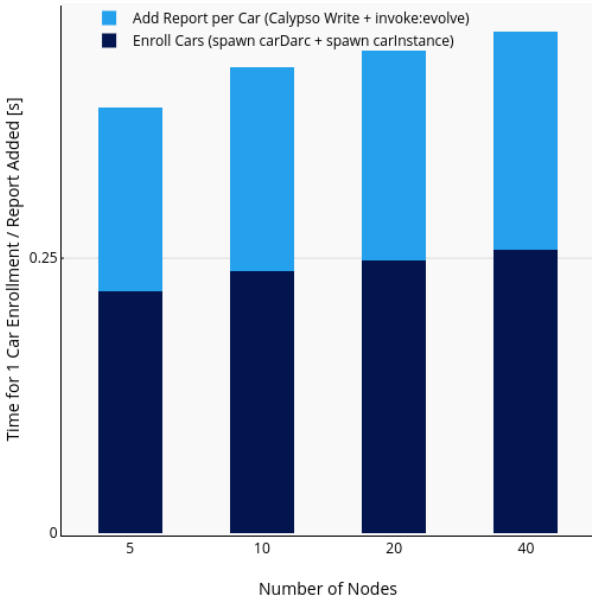


Figure 19: Wall Time for a Car Enrollment/Report Added with Various Number of Nodes

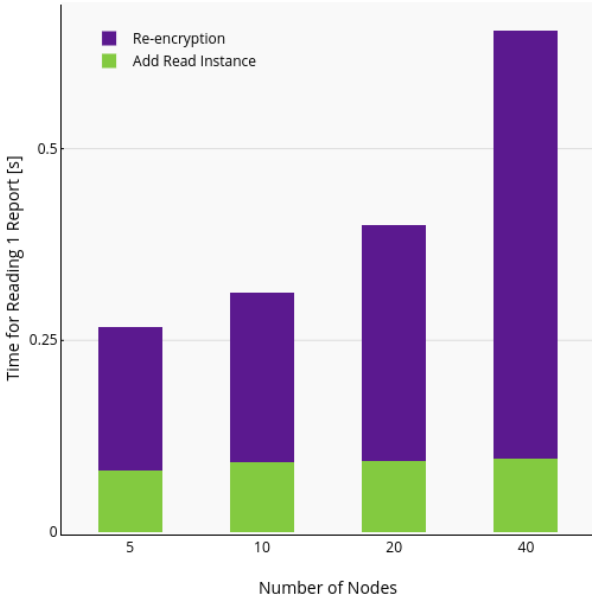


Figure 20: Wall Time for Reading a Report with Various Number of Nodes

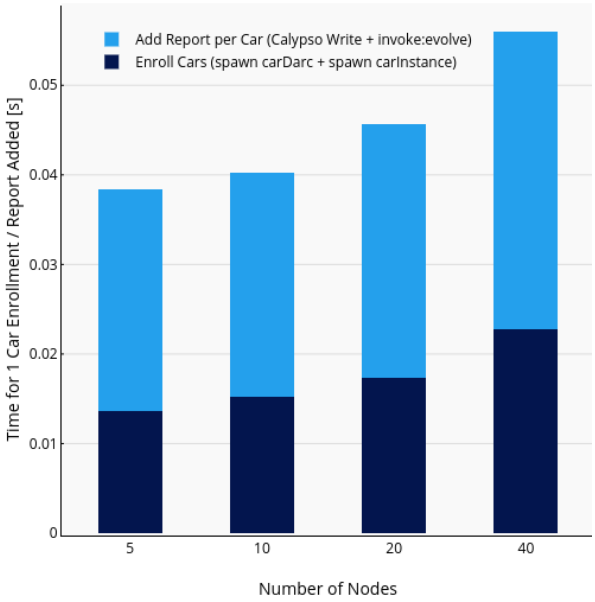


Figure 21: System Cost for a Car Enrollment/Report Added with Various Number of N

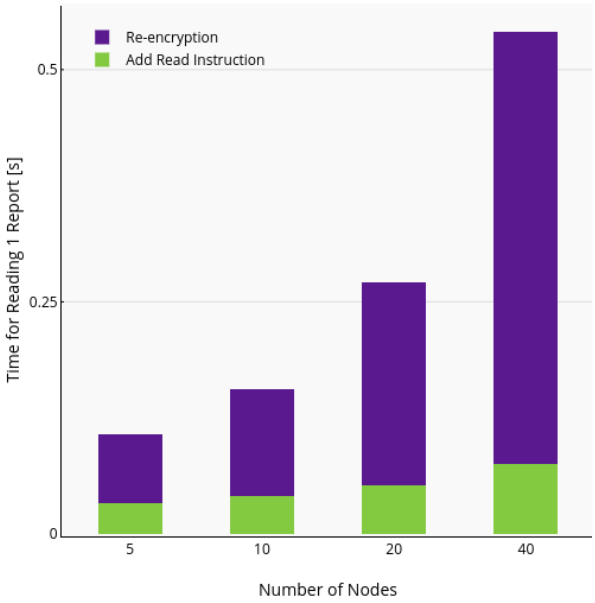


Figure 22: System Cost for Reading a Report with Various Number of Nodes

# 6 Conclusion and Future Work

We have presented a decentralized solution for fighting frauds in the automotive industry and establishing trust between vehicle owners, potential buyers, car manufacturers, garages, insurance companies and car dealers.

In our work, we designed a service using the ByzCoin [15] blockchain protocol together with the Calypso [7] framework for securely storing private data on a blockchain.

Overall, our implementation builds on top of the Cothority Template [11] (a starting point for creating a ByzCoin service) and includes definition of the access control structure, creation of the necessary car contract, handling the sensitive data with Calypso and development of a desktop Java application, used for interaction between the users and the blockchain service.

After producing a prototype, we have tested the correctness with unit and integration tests. Once we were confident that it is a valid proof of concept, we proceeded with stressing the network by introducing more nodes and concurrent transactions.

The evaluation has demonstrated that our system works well with up to 500 car enrollments in parallel. Moreover, with constant number of hosts, the wall time (number of seconds it takes in real life, including the network communication) calculated per transaction is greater when there are lower number of concurrent transactions, because the block creation time is equal in every scenario, whereas the system cost per transaction remains approximately the same independently on the number of concurrent transactions. Furthermore, when we modified the number of hosts, we observed that it takes longer to communicate with the Calypso secret management cothority, due to the time used for coordination between nodes.

As a possible future task, we consider expanding the system to work on a larger scale (for example vehicles from the entire world). The approach that we would follow is **sharding** of the nodes, introduced in the OmniLedger paper [22]. It increases the transaction processing capacity with the addition of new members to the network [22], as it does not require each one of them to validate every transaction in the system. Therefore, it also reduces the load of the nodes. Additionally, we propose a future expansion that includes creation of transactions directly by IoT devices attached to the vehicles.

With this proof of concept, we showed one of the many possible applications of the blockchain technology for establishing trust among distributed, distrustful parties.

# 7 Acknowledgements

I would like to thank Prof. Bryan Ford and Alexandru Rusu for providing me with the opportunity to work on a project of my interest.
Additionally, special thanks to my supervisor Linus Gasser for his guidance and help throughout the whole project.

## References


[1] NAKAMOTO, S. Bitcoin: A Peer-to-Peer Electronic Cash System, 2008.

[2] LINN, L. A., AND KOO, M. B. Blockchain for health data and its potential use in health it and health care related research. In ONC/NIST Use of Blockchain for Healthcare and Research Workshop. Gaithersburg, Maryland, United States: ONC/NIST (2016)

[3] JUAN DELACRUZ, I. B. Blockchain is tackling the challenge of data sharing in government, May 2018

[4] DEDIS, ByzCoin, accessed 11.02.2019, <`https://github.com/dedis/cothority/blob/master/byzcoin/README.md`>

[5] DEDIS, ByzCoin Figure, accessed 08.03.2019, <`https://raw.githubusercontent.com/dedis/cothority/master/byzcoin/ByzCoin.png`>

[6] DEDIS, Calypso, accessed 12.02.2019, <`https://github.com/dedis/cothority/tree/master/calypso`>

[7] E. Kokoris-Kogias, E. Ceyhun Alp, S. Deepthy Siby, N. Gailly, L. Gasser, P. Jovanovic, E. Syta, and B. Ford. CALYPSO: Auditable Sharing of Private Data over Blockchains, 2018

[8] NIKITIN, K., KOKORIS-KOGIAS, E., JOVANOVIC, P., GAILLY, N., GASSER, L., KHOFFI, I., CAPPOS, J., AND FORD, B. CHAINIAC: Proactive Software-Update Transparency via Collectively Signed Skipchains and Verified Builds. In 26th USENIX Security Symposium (USENIX Security 17) (2017), USENIX Association, pp. 1271âĂŞ1287.

[9] DEDIS, Contracts, accessed 11.02.2019, <`https://github.com/dedis/cothority/blob/master/byzcoin/Contracts.md`>

[10] DEDIS, Cothority, accessed 11.02.2019, <`https://github.com/dedis/cothority`>

[11] DEDIS, Cothority Template, accessed 30.09.2018, <`https://github.com/dedis/cothority_template`>

[12] DEDIS, Darc, accessed 06.02.2019, <`https://github.com/dedis/cothority/tree/master/darc`>

[13] J. Ian Munro, Thomas Papadakis, and Robert Sedgewick. Deterministic Skip Lists. In Proceedings of the Third Annual ACM-SIAM Symposium on Discrete Algorithms, SODA âĂŹ92, pages 367âĂŞ375, 14 Philadelphia, PA, USA, 1992. Society for Industrial and Applied Mathematics

[14] KARAME, G. O., ANDROULAKI, E., AND CAPKUN, S. Double-spending fast payments in Bitcoin. In 19th ACM Conference on Computer and communications security (2012), ACM

[15] E. Kokoris-Kogias, P. Jovanovic, N. Gailly, I. Khoffi, L. Gasser, and B. Ford. Enhancing Bitcoin Security and Performance with Strong Consistency via Collective Signing. In Proceedings of the 25th USENIX Conference on Security Symposium, 2016

[16] WOOD, G. Ethereum: A Secure Decentralised Generalised Transaction Ledger. Ethereum Project Yellow Paper (2014)

[17] European Parliament, created 29.05.2018, accessed 05.02.2019, <http://www.europarl.europa.eu/news/en/headlines/society/20180525STO04312/fighting-mileage-fraud-on-used-cars>

[18] SYTA, E., TAMAS, I., VISHER, D., WOLINSKY, D. I., L., GAILLY, N., KHOFFI, I., AND FORD, B. Keeping Authorities âĂIJHonest or BustâĂİ with Decentralized Witness Cosigning. In 37th IEEE Symposium on Security and Privacy (May 2016)

[19] EYAL, I., AND SIRER, E. G. Majority is not enough: Bitcoin mining is vulnerable. In Financial Cryptography and Data Security. Springer, 2014

[20] AZARIA, A., EKBLAW, A., VIEIRA, T., AND LIPPMAN, A. Medrec: Using blockchain for medical data access and permission management. In Open and Big Data (OBD), International Conference on (2016), IEEE, pp. 25âĂŞ30

[21] DEDIS, Mininet, accessed 25.02.2019, <https://github.com/dedis/onet/blob/master/simul/platform/MININET.md>

[22] KOKORIS-KOGIAS, E., JOVANOVIC, P., GASSER, L., GAILLY, N., SYTA, E., AND FORD, B. OmniLedger: A Secure, Scale-Out, Decentralized Ledger via Sharding. In Security and Privacy (SP), 2018 IEEE Symposium on (2018), Ieee

[23] DEDIS, Onet Simulation, accessed 25.02.2019, <https://godoc.org/github.com/dedis/onet#Simulation>

[24] M. Castro and B. Liskov. Practical Byzantine Fault Tolerance. In 3rd USENIX Symposium on Operating Systems Design and Implementation (OSDI), Feb. 1999

[25] Google, Protocol Buffers, accessed 15.02.2019, <https://developers.google.com/protocol-buffers/>

[26] R.C. Merkle, "Protocols for public key cryptosystems," In Proc. 1980 Symposium on Security and Privacy, IEEE Computer Society, pages 122-133, April 1980

[27] DEDIS, Simulation, accessed 25.02.2019, <https://github.com/dedis/cothority_template/tree/master/simulation>

[28] DEDIS, Skipchain Explorer, accessed 24.02.2019, <https://github.com/dedis/student_18_explorer>

# 8 Apendix

## 8.1 Step-by-step Guide for Installation

The bellow mentioned steps need to be taken into consideration when setting the system up:

1. Download and install the Go programming language distribution (`https://golang.org/doc/install`)
2. Download and install Java and Maven (`https://www.oracle.com/technetwork/java/javase/downloads/index.html`)
3. Download and install Docker (`https://runnable.com/docker/getting-started/`)
4. Optional but recommended:
   Download and install IDE for Developers that supports Go, e.g. IntelliJ IDEA
   (`https://www.jetbrains.com/idea/download` with Go plugin)
5. Run the following commands in terminal to download the project and required dependencies:

```
$ go get github.com/dedis/cothority
(and switch to the byzgen_1810 branch)

$ go get github.com/dedis/student_18_car

$ go get -d ./... (in student_18_car directory)
```

6. Create Docker Container for the conodes and start them with the following commands:

```
$ make docker (in student_18_car directory)

$ docker run -p 7002-7009:7002-7009 --name car
-ti dedis/conode_template-test
```

7. Run Main.java which is located in student_18_car/external/java/src/main/java/ch/epfl/dedis/template/gui/index in order to start the client application

## 8.2 Repository

- `https://github.com/dedis/student_18_car`