Cluster Computing (2020) 23:2105–2122
https://doi.org/10.1007/s10586-020-03063-9

HRM smart contracts on the blockchain: emulated vs native

Ray Neiheiser[1] · Gustavo Inácio[2] · Luciana Rech[1] · Joni Fraga[2]

Received: 27 August 2019 / Revised: 23 January 2020 / Accepted: 28 January 2020 / Published online: 6 February 2020

Abstract
In the past years numerous scandals in hiring processes of public institutions revealed significant weaknesses of the existing process. This lead to a lose of trust of citizens and applicants in these processes. Besides that, even honest processes often lack transparency for applicants, the company and, in the case of public institutions, also for citizens. Distributed ledger technology has been used in numerous past projects to establish trust between entities. Especially smart contracts have been a useful tool to execute programs in this setting. Thus, in the context of this project we developed an approach based on smart contracts to decentralize this process and improve its transparency and reliability. We enhance the system with game theory mechanics to encourage reviewers and candidates to participate honestly, further increasing the likelihood of a fair selection process. Nonetheless, not all blockchains support smart contracts and their usage comes with an elevated additional monetary cost. Due to that, in the blockchain environment, many businesses only emulate smart contracts by executing them on external servers which improves the scalability but decreases the decentralization. In the context of this, we compare the advantages and disadvantages of emulating smart contracts compared to native smart contracts in our concrete usage example. Our approach, compared to existing solutions, can be employed on any distributed ledger which allows to store some sort of metadata on the chain resulting in a significantly lower creation and maintenance cost (up to 30 times cheaper). The developed solution and discussion gives useful insight into safety and cost considerations which have to be made when deciding between native and emulated smart contracts.

Keywords Blockchain · Smart-contracts · Distributed ledgers · Transparency

1 Introduction

In each company or public institution there is a sector called *Human Resource Management* (HRM) which is responsible for hiring new employees. In past studies, it has been shown, that how well this organ performs has a significant impact on the overall performance of the company or institution [5, 17].

In this context, it is very important for the company as well as for the applicants that this process is as transparent as possible. For the company it is important to guarantee that the most qualified applicant is selected and for the applicants it is crucial to know for which reason they passed to the next phase or not, which enables them to improve these points for future selection processes. Besides that, in the context of public institutions, these processes have to be especially transparent for the citizens of the corresponding country to guarantee a fair selection process for all applicants and to avoid corruption. Unfortunately, in practice, in most cases these processes lack transparency since they rely on trust into a single central department (the HRM).

In Brazil, where public institutions often offer excellent salaries and stability they, therefore, easily attract corrupt agents competing with honest applicants, this has become a huge problem. As an example of this, there were countless

✉ Ray Neiheiser
ray.neiheiser@gmail.com

Gustavo Inácio
inacio.gusta@gmail.com

Luciana Rech
luciana.rech@ufsc.br

Joni Fraga
joni.fraga@ufsc.br

[1] Departamento de Automação e Sistemas, Universidade Federal de Santa Catarina (UFSC), Florianópolis, Brazil

[2] Departamento de Informática e Estatística, Universidade Federal de Santa Catarina (UFSC), Florianópolis, Brazil

scandals of manipulation during these processes in Brazil. In 2017, in one operation of the federal police, 98 selection processes had been found manipulated [20, 27]. More recently, the police discovered the corruption of several processes involving multiple municipals [18, 19] over several years. Nevertheless, this is not an occurrence which is exclusive to Brazil or to public institutions.

We, therefore, propose a more transparent and decentralized solution based on the Distributed Ledger Technology (DLT). The DLT, which had initially been developed to solve the double spending problem, has been applied to solve a wide range of problems in the fields of economy, sociology, politics, biology, and many more [2, 8, 37]. The blockchain, which is the most popular storage module of the DLT, consists of blocks which are chained together by including the hash of the previous block in every subsequent block. This is usually combined with an algorithm which establishes byzantine fault tolerant consensus to build the distributed ledger [23].

Instead of relying on a centralized system, which requires trust, the distributed ledger removes this necessity and distributes the application over several nodes.

In the context of this, smart contacts have been developed, which are self-executable programs (code) which are stored in a distributed ledger for future invocation. This allows, for example the creation of escrow mechanisms which do not require any trusted third party since the contract is executed automatically on all nodes when certain criteria are met. Some distributed ledgers allow any user to create smart contracts and store them on the blockchain (as in Ethereum). This kind of contract we call native smart contracts in the context of this work, other more specialized distributed ledgers have smart contracts embedded into the base code of the blockchain and do not allow to add additional smart contracts (Steem is an example of that). In the case of these blockchains, more flexible smart contract functionality can only be emulated on an external server.

Several start-up companies propose solutions to decentralize the job market and facilitate the search of employees for companies using DLT. Most of these solutions run on specialized blockchains with embedded smart contracts. Examples of this are Ouna [26] or Cverification [10]. But, while many of these proposals verify the curriculum by storing diplomas and certificates on the chain, most of these approaches rely on the universities or previous employers to sign the data to certify its validity. Additionally, most existing proposals neither include mechanisms to protect the privacy of the applicant nor improve the transparency of the process.

By executing the selection process using smart contracts on the blockchain, as explained in our previous paper [24] we decentralize the selection process in a manner that guarantees transparency and protects both the applicant and the selecting institution from malicious actions.

Nonetheless, executing the smart contract code is expensive since they have to be executed deterministically by all servers participating in the distributed ledger, especially considering that all invocations have to be executed in the same order. Due to this, on most platforms (as *Ethereum*) the number of smart contracts the system can execute is computationally bound leading to a bottleneck. Some works, like Chainspace [1], optimize this by dividing the distributed ledger into partitions (clusters) where each smart contract is only executed in one of the partitions. Thus, allowing to execute several different smart contracts in parallel. While this increases the performance significantly, it decreases the decentralization due to the reduced number of servers participating in the contract execution and verification. Besides that, while solving the scalability problem, the volatility of the execution cost, depending on the load of the system, is still a problem. As visible in the Ethereum gas price history in [13] the price to invoke or to create a smart contract often spikes to more than 100 times the average price.

Due to that, many blockchain applications nowadays only emulate smart contracts to reduce the complexity and reduce the execution cost. Examples of this are many popular applications like *Splinterlands*, *Next Colony* or *Actifit* which are built on top of popular distributed ledgers without native support for smart contract (like Steem [32]). *Emulated* smart contracts work similar to native smart contracts but are only executed on a restricted number of dedicated servers outside of the distributed ledger. In this case, a single server or a cluster of servers is created which polls new blocks from the blockchain and then reacts accordingly to certain invocations. While this may centralize the execution, all the invocations and their results are still stored on the blockchain which maintains the level of transparency of native smart contracts. Besides that, if the code of the *emulated* smart contracts is open source it is easy to verify the correctness of the execution of the contract maintaining a similarly high trust factor. In this case, if no exchange of assets is involved, the only disadvantage is the maintenance of external server infrastructure.

In this paper we propose and evaluate *emulated* HRM smart contracts and compare it with native implementations in terms of cost but also in terms of security implications. This is particularly interesting for anyone who wants to build an application on the blockchain but might not need to harness the full decentralization of smart contracts and thus also should not have to bear the full costs of them.

Based on this our paper makes the following core contributions.

– We show a more detailed description of the model we proposed in [24].
– We rework this approach to support emulated smart contracts which allows to execute it on any blockchain;
– We compare native smart contracts with emulated smart contracts in terms of cost, usage and safety implications in the light of this usage scenario.

The next section explains the background knowledge which is necessary for the comprehension of this paper. Following that we head into the proposals including the native as well as the *emulated* approach. After that, we compare the performance of both approaches before we head into a brief discussion about the implications of both approaches. Then, we finish with the related work and finally, with the conclusion.

2 Theoretical background

This section aims at elaborating the theoretical background knowledge which is necessary for the comprehension of this paper. In this context, we explain several details surrounding distributed ledgers, the blockchain and smart contracts. Besides that, we also present a case study of a traditional selection process on which we based our model and prototype.

2.1 Blockchain

As displayed in Fig. 1, the blockchain is similar to a linked list where each node is connected to the previous node by its hash. This way, when trying to change any part of any of the elements in the list, all following nodes had to be adapted to include the new hash. This makes it computationally expensive and difficult to manipulate past blocks since all following blocks had to be adapted as well. Thus, the blockchain is defined as immutable.

While most blockchains are public (*permissionless*), where anyone can participate in the consensus, there also are consortium and private blockchains (*permissioned*) with higher privacy guarantees but access, transparency and decentralization are limited.

Most public blockchains rely on Game Theory for correctness by paying an incentive to reward processes for honest participation to make it more worthwhile to participate honestly in the system than trying to corrupt it. In this context, the replicas which confirm the validity of the transaction and participate in the costly consensus (called miners) receive an incentive in cryptocurrency [23] for doing so.

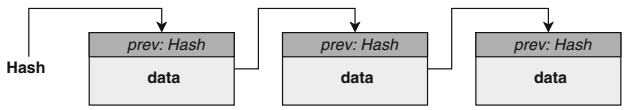

Fig. 1 The blockchain—Blocks connected by the hash of the previous block

As noted earlier, the blockchain is mainly known for its application in distributed ledgers. A distributed ledger consists of a replicated blockchain on several sites. A consensus is required in order to guarantee that all sites, or at least the majority of them, hold the same copy of the blockchain. Since most ledgers are public, which means anyone can launch a server to participate in the consensus, byzantine failures have to be tolerated. These special failures englobe arbitrary behaviour as bit flips but also intentional malicious behavior and collusion. Traditional Byzantine consensus requires a majority of $3f + 1$ out of n replicas to guarantee safety. Nevertheless, since most distributed ledgers as, for example, Bitcoin [23] or Ethereum [36] are public, anyone could deploy multiple replicas to change the consensus to his favor.

For this reason, Proof of Work (PoW) had been developed where the participants in the consensus (also referred to as miners) have to solve a cryptography riddle in order to propose a block. This transforms the consensus from *one process/one vote* to *one CPU/one vote* making it essentially more difficult and costly to influence the consensus maliciously. Nevertheless, also with PoW, at least 50% of all replicas have to be correct, and, as some recent research showed, some Byzantine attacks as *selfish mining*[1] may be effective with 25% computing power already [35].

Clients within the blockchain are identified by a key-pair consisting of a public and a private key where the public key serves as its identity, establishing pseudo-anonymity.

In the typical work-flow a client submits a transaction to several nodes in the network, nodes verify the transaction and then include it in the next block they are building. After some period of time or when the block is full, the replica tries to solve a computational riddle. If the replica was the first to solve the riddle the majority of the other replicas will receive its block, check for its validity and if valid append it to their blockchain. Since this process is optimistic, there is a chance of two servers creating and distributing a block in the exact same moment. In this case, parts of the ledger can theoretically have different blockchains. Nevertheless, depending on which replica solves the next riddle, it will force the other replicas to accept its view of the blockchain. This way it is guaranteed that eventually the right order of blocks is figured out and a

[1] Selfish mining describes an attack scenario where an attacker forks a blockchain deliberately and continues mining on its own fork until it has the longer queue of blocks and then imposes its own view of the ledger on the remaining consensus replicas.

majority of the participating servers agree on the same state. Other models exist, as for example Proof of Stake (PoS) where the stake of the owner of the replica in the ledger is considered this reduces the computational costs significantly since servers with higher stakes in the system also have less interest in corrupting it, thus the order of the servers is often decided deterministically based on the stake of the servers. Besides that, approaches based on traditional Byzantine fault tolerance exists in, for example, *Hyperledger* [6]. Nevertheless, these follow a permissioned model where the entry or exit of consensus nodes requires consensus which improves the overall performance but restricts the decentralization (since less independent parties are going to be responsible for guaranteeing the safety of the consensus).

2.2 Smart contracts

In the context of blockchains, smart contracts had been developed by Szabo [33], these smart contracts can be deployed on the blockchain and executed similarly to replicated state machines. This allows deploying state machines dynamically in a replicated setting extending the application possibilities of blockchains over the traditional use of peer-to-peer asset transfer. Smart contracts have been used in several works to implement escrow mechanisms, games, digital identity management or even to replace traditional contracts. Since we are dealing with a replicated state machine all smart contracts must execute the same code at all locations deterministically. At the same time, it is important that the code finishes in finite time which is an essential property of SMR (State Machine Replication). This means that smart contracts have to be deterministic and Turing complete. For example, ledgers like Ethereum guarantee this by creating an artificial language which only allows deterministic code and by applying a cost on the execution of a transaction. This cost is often referred to as *gas*. Every transaction invoking a smart contract costs a certain quantity of *gas* which has to be sent together in the transaction which creates or invokes a smart contract. If the contract runs out of *gas* during executing time it will be rolled back and the *gas* is lost. Since *gas* has monetary value this motivates the creator of the smart contract to limit the required computation time and memory usage. Other systems, as Hyperledger, rely on an execute-order protocol similar to deferred update [31], where the smart contract (called chaincode) is first invoked on local servers and the results are then sent to the consensus protocol to guarantee that all replicas receive the same result in the same order. The local servers only stop running the chaincode if a certain time-out is reached. Since only a limited number of replicas execute the contract and since the execution of independent contracts can be on different threads, it is difficult to affect the system liveness and safety by sending non deterministic chaincode. Thus, chaincode can be written in a number of different languages.

2.3 Public selection processes in Brazil

According to the Brazilian Law 8.112 section 3 article 11, a public selection process consists of exams or exams and evaluations of titles. These can be executed in two phases where the candidate applies by paying an alleged amount specified in a notice of the vacancy.

Each selection process must define a statute which introduces all necessary details and rules related to the vacancy and selection process. As an example we analyze the statute 053/2018/DDP/UFSC from 05/07/2018 of the Federal University of Santa Catarina defining the following steps: One written exam, the verification of titles and publications, and one oral presentation.

In the context of this, first a committee of the local university has to be created which is then responsible for installing a board of professors including at least one external professor responsible of evaluating the candidates. The task of this board is to evaluate the candidates. Depending on the number of candidates the selection process can take several months, including the publication of the statute, registering of the applicants, elaboration, supervision and correction of the exams.

Analyzing this selection process, the complexity and time consumption for the participating professionals of the existing process, besides several central points of failure, become apparent. For example, a possibly corrupt committee could easily instantiate a biased board of professors which in turn can easily make biased decisions regarding a candidate. Following the issues described above, it is clearly visible that there is a need for a more transparent and decentralized selection process. Our proposals to solve these issues are described in the next Section.

3 Proposals

As elaborated earlier, in this paper, we propose two different approaches to solve the previously discussed issues of selection processes. Unlike existing approaches, our solutions are blockchain agnostic which means that a prototype can be developed for any existing distributed ledger. While both processes follow a similar scheme, the first model, also discussed in [24] proposed an approach and architecture which can be applied to any blockchain that supports smart contracts and has a built-in cryptocurrency. These premises are necessary since we use smart contracts for execution and automation and the

cryptocurrency as an incentive in that model. The second model runs completely blockchain agnostic, independently if the blockchain offers smart contract functionality or not, nonetheless, we still require some sort of currency as an incentive. We assume that malicious adversaries may control up to f out of $n = 3f + 1$ of the reviewers and similarly a maximum of f out of $n = 3f + 1$ of the blockchain nodes. These adversaries can collude but we assume they don't have sufficient computing power to compromise the cryptographic principles of the blockchain.

In any of the two proposals, we assume three types of entities which can be identified by their public key in the system: the applicant, the reviewer and the institution.

3.1 Smart contracts of the proposals

Both proposals use the abstraction of smart contracts. Nonetheless, the first proposal runs them natively as actual smart contracts on the blockchain and the second proposal runs the code separately on a dedicated server. Thus, the general approach of the two proposals is very similar, while principally the invocation dynamic and setup changes.

Following that, there are two types of smart contracts in the system. There is the *List of Institutions* smart contract which holds a list of participating institutions which itself contains a list of viable reviewers. And the *Vacancy Contract* which holds vacancy information and can be invoked to register for a vacancy or to execute the phases of the selection process.

The proposed system shows several advantages compared to the existing system:

- Increased transparency: by showing the results on the blockchain after every application step.
- Pseudo-anonymity of the evaluating reviewers and applicants.
- Smart contracts are context-free: with no room for speculation as they are defined in a context-free language (Opposed to contract in natural languages).
- Increased trust: by decentralizing the evaluation to random reviewers of different institutions to decrease corruption.
- It decreases the amount of manual work required of the participating reviewers by automatizing and distributing the process.

3.1.1 Institution list contract

The *Institution List* contract consists of a list of institutions with a common interest (a *consortium*). New institutions can be included (registered), but require a certain quantity of existing institutions to accept the new member (consensus). As noted earlier, an institution can be identified by its public key, which, for example, can be included in the browser certificate of the institutions home page to allow easy identification.

This constructs a semi-permissioned model on top of the permissionless blockchain allowing to maintain the privacy of the applicants since only reviewers from selected institutions will have access to the private data of the applicants.

Additionally, each institution introduces in this contract a list of reviewers which the institution can register and unregister freely. The *Institution List Contract* is invoked by the *Vacancy Contract* (explained in the next subsection) to obtain a random selection of reviewers of different institutions. To guarantee reproducibility and to make sure that the random selection is deterministic, a random seed based on block parameters as the block hash is chosen. Since this block hash is not previously known it is difficult to manipulate or maliciously influence this process.

3.1.2 Vacancy contract

The *Vacancy Contract* which is created by the institution itself to include all the necessary requirements of a given selection process, is the only contract the applicant and reviewers will have to interact with. Following the selection process outlined in Sect. 2.3, an applicant may register with the contract until a certain deadline is met, afterwards the *Vacancy Contract* invokes the corresponding institution list contract to obtain the list of random reviewers. Then, the applicants must provide the necessary data for each step of the selection process. The reviewers, on the other hand, ought to provide the evaluations for each applicant until a certain deadline is met and will finally be rewarded for their participation in the process. After the application process is finished, the resulting data can be obtained transparently from the blockchain.

3.2 Native smart contracts

This subsection gives a detailed technical overview of the proposal using native smart contracts as proposed in [24].

3.2.1 Institution list

Following the execution scheme shown in Fig. 2, the first step of the protocol is the creation of the smart contract by any institution (step 1). This involves writing the smart contract in the language supported by the platform (i.e: Solidity in the case of our prototype) and submitting the code in a signed transaction to the blockchain (step 2). Depending on the platform the transaction will cost a certain quantity of currency.

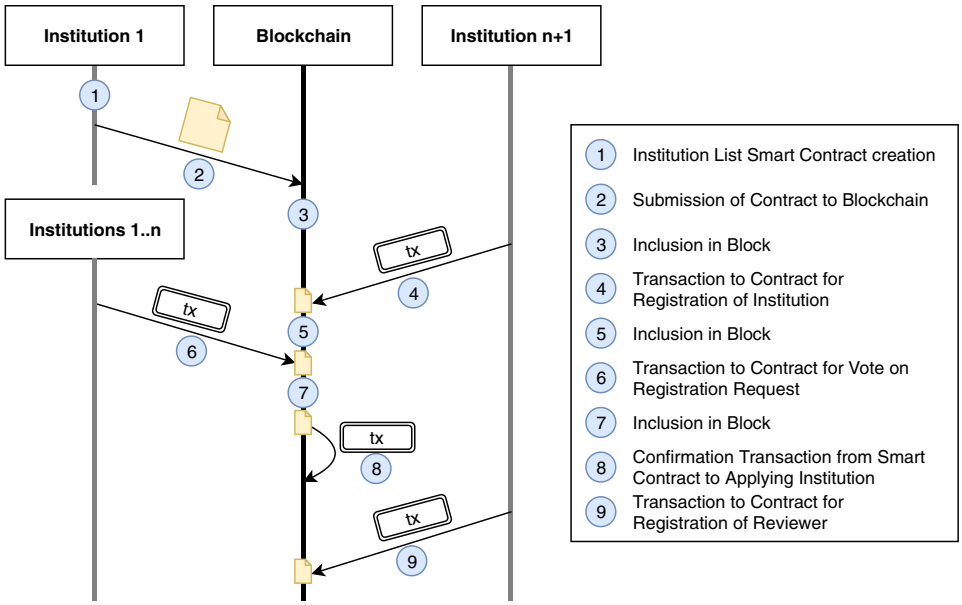

Fig. 2 Institution list smart contract dynamic

If successfully deployed (step 3), the server to which the transaction was submitted, will return an acknowledgement of the inclusion in a block. All successfully included transactions return an acknowledgement to the sender. Nonetheless, since there is a probability of forks (as explained earlier) the transaction is only guaranteed with a high probability after several blocks passed (Most Proof of Work blockchains do not offer finality). Thus, to simplify the figures of the interaction with the blockchain we neither display the transaction acknowledgements nor the waiting time.

After that, at any moment, in step 4, where n determines the total number of already registered institutions in the list, any institution $(n + 1)$, can request to be registered by sending a signed transaction to the address of the smart contract on the blockchain. Then, in step 5, the transaction gets included in the blockchain which invokes the contract. As a result of this, the previous n institutions have the opportunity to vote on the request (step 6) which also, eventually, get included in the blockchain in step 7. After a majority of votes has been done, the last vote will trigger a transaction back to institution $n + 1$ (step 8) notifying the institution about the result. If successful, the institution can now send transactions to the contract to start registering reviewers (step 9).

3.2.2 Vacancy contract

The initial dynamic of the *Vacancy Contract* is very similar, as depicted in Fig. 3. In the first step the institution creates the *Vacancy Smart Contract* based on the requirements of the process including deadlines, number of phases, etc. This then gets sent in the second step in a signed transaction to the blockchain and then, eventually, included in a block (step 3). After included in the block, applicants are able to request their registration by sending a certain quantity of currency in a signed transaction to the address of the smart contract on the blockchain (step 4). If successful, this transaction also gets included in the blockchain (step 5). Then, after a certain defined timeout (registration deadline at step 6), the smart contract stops accepting new registrations and the next transaction (step 7) addressed at the contract (from any party) will trigger the institution list to obtain a random list of reviewers (step 9). In this case, the *Vacancy Contract* executes a read request to obtain a certain quantity of random reviewers from the list of institutions. It depends on the *Vacancy Contract* to determinate how many reviewers to fetch. Based on on the quantity of available reviewers in the list of institutions the request can include details as the area of expertise but also exclude certain reviewers due to a conflict of interest with some candidates. The request also includes the random seed used for the execution to guarantee a deterministic result on all servers.

In Fig. 4 the typical process of the application phase is then displayed. In the first step, applicants store their personal data, encrypted with the public keys of the reviewers on a server outside of the scope of the blockchain. By encrypting it with the public key of the reviewer it is guaranteed that the data cannot be read by any unauthorized party. The public keys of the reviewers are their identifiers on the blockchain and thus can be easily obtained. In the next step (step 2), the applicant then stores the hash of the data on the blockchain by sending a signed transaction to the smart contract to guarantee that all reviewers receive the same dataset from the applicant.

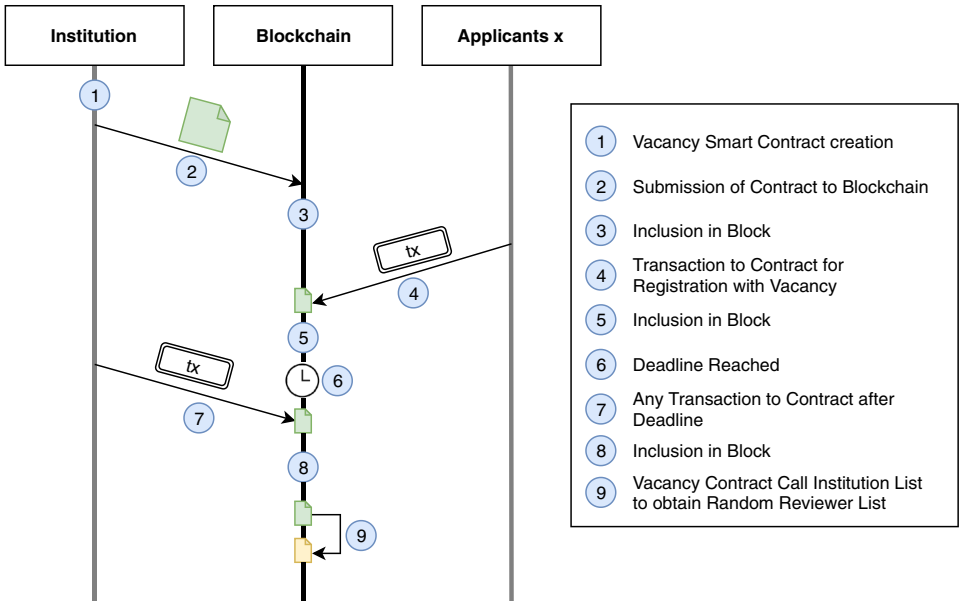

Fig. 3 Vacancy smart contract registration dynamic

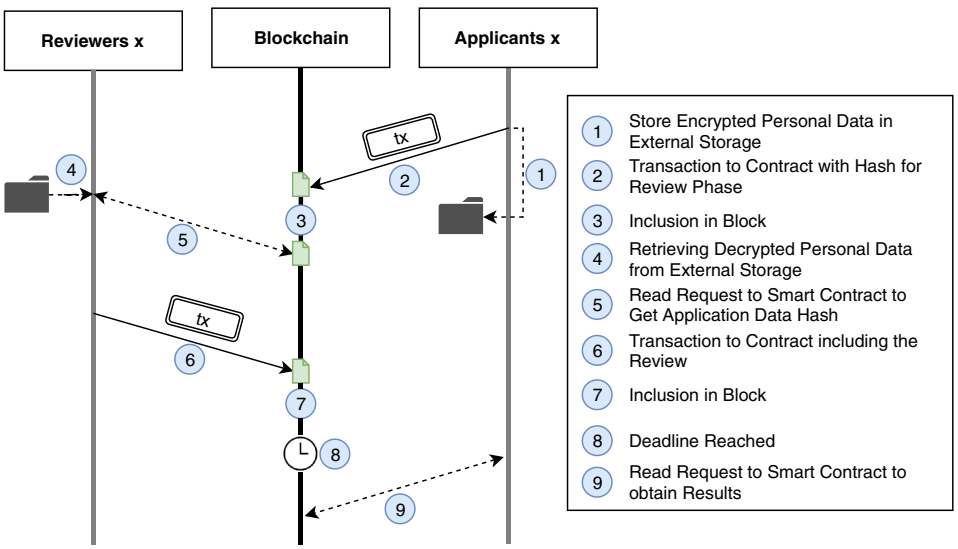

Fig. 4 Vacancy smart contract application phase dynamic

After being included on the blockchain (step 3), the reviewers will fetch the user data from the private repository in step 4 and then obtain the hash the applicant stored from the smart contract. Since it is merely a read request to one of the nodes, no transaction is required and, thus, also no consensus on the blockchain. After that the reviewer will compare the hash from the blockchain with the hash of the data he obtained from the private repository, evaluate the data and send back a transaction with the review to the blockchain (step 6). This is then included in the blockchain in step 7 until a certain deadline is reached again (step 8). After the deadline any user can send read requests to obtain the results from the application phase (step 9).

At the end of the application process, a part of the currency the users sent for the application process is returned to the users if they received a higher grade than a given threshold to encourage the user to send valid data. At the same time, another share of the currency is rewarded to the reviewers which came to similar conclusions as the other reviewers as an incentive to review the applications honestly and thoroughly.

Each vacancy may consist of several application phases for which different reviewers may be chosen. This process depends on the necessities of the institution. Different reviewers could also be chosen for each user independently, resulting in a higher execution cost.

3.3 Emulated smart contracts

This section describes the approach which emulates smart contracts on a common blockchain which does not offer smart contract support. The proposed approach is very similar to the native contracts but requires some additional setup.

3.3.1 Emulated institution list contract

Figure 5 shows the dynamic to set up the emulated smart contract for the institution list. In the first step, a server has to be setup to be able to run the smart contract code. Then, in the second step, similarly to the native smart contracts, the code of the *Institution List Contract* has to be created. But, different to the native smart contracts. this contract can be written in any programming language but, additionally code has to be created that regularly polls updates from the blockchain to detect invocation of the smart contracts. Besides that, in the code of the contract the public key to invoke the smart contract has to be defined (Thus a wallet for this has to be created). In the next step, this code is then sent and deployed on the dedicated server.

Following that (in step 5), a transaction containing the keyword "create" with the destination address (as specified in the contract code) has to be sent to the blockchain. After included in the next block (step 6), through regular polling (step 7), the contract detects the invocation to the specified contract, creates a new instance of the contract and issues a confirmation transaction to the blockchain with the *id* of the new instance (step 9). This allows to run several independent instances of the emulated smart contract using the same server. Any instance can be individually invoked by sending a transaction to the defined address with the resulting id.

Figure 6 shows the dynamic of the emulated *Institution List Contract*. For the institutions it stays quite similar to the native smart contract. But, different to native smart contracts each transaction not only includes the specific address of the contract but also contains the id of the specific instance to be invoked and name of the operation. In the first step, a new institution $(n + 1)$ sends a transaction to the specified address, to the ledger, with the specified id. After this has been included in the blockchain, in step 2, the smart contract script on the dedicated server detects the invocation (steps 3–4) and answers with a transaction to the blockchain confirming the registration (step 5). Then, this last transaction is included in the next block and any of the existing institutions 1, 2, ...*n* can send a transaction consisting the keyword "vote" and the public key of the institution they want to vote, to the ledger, to vote on the entry of any new institution (step 7). Again, this has to be included in the blockchain, polled and detected by the emulated smart contract and each time a confirmation transaction is created (steps 8–11). After the last necessary vote, the contract will issue a transaction to the blockchain to confirm the result of the consensus about the inclusion into the institution list (also step 11). If successful, the institution $n + 1$ can then start registering its reviewers with the contract (step 12), by including the "registerReviewer" keyword which again have to go through the steps of inclusion in the blockchain, polling and detection (steps 13–15).

3.3.2 Emulated vacancy contract

Similar to the creation of the *Institution List*, also the *Vacancy Contract* has to be setup as shown in Fig. 7. This starts with the setup of the dedicated server (where the same server of the *Institution List* can be reused) in the first

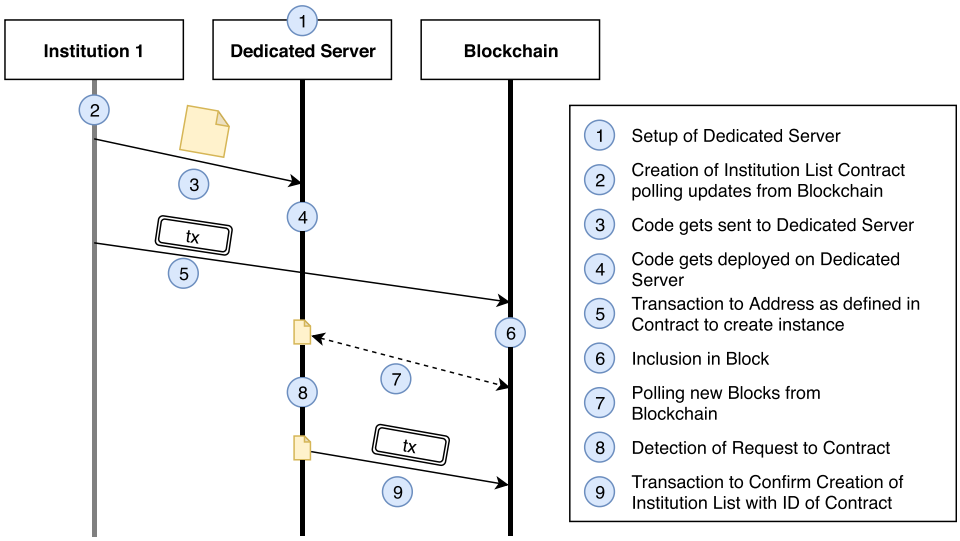

Fig. 5 Creation of the emulated institution list smart contract

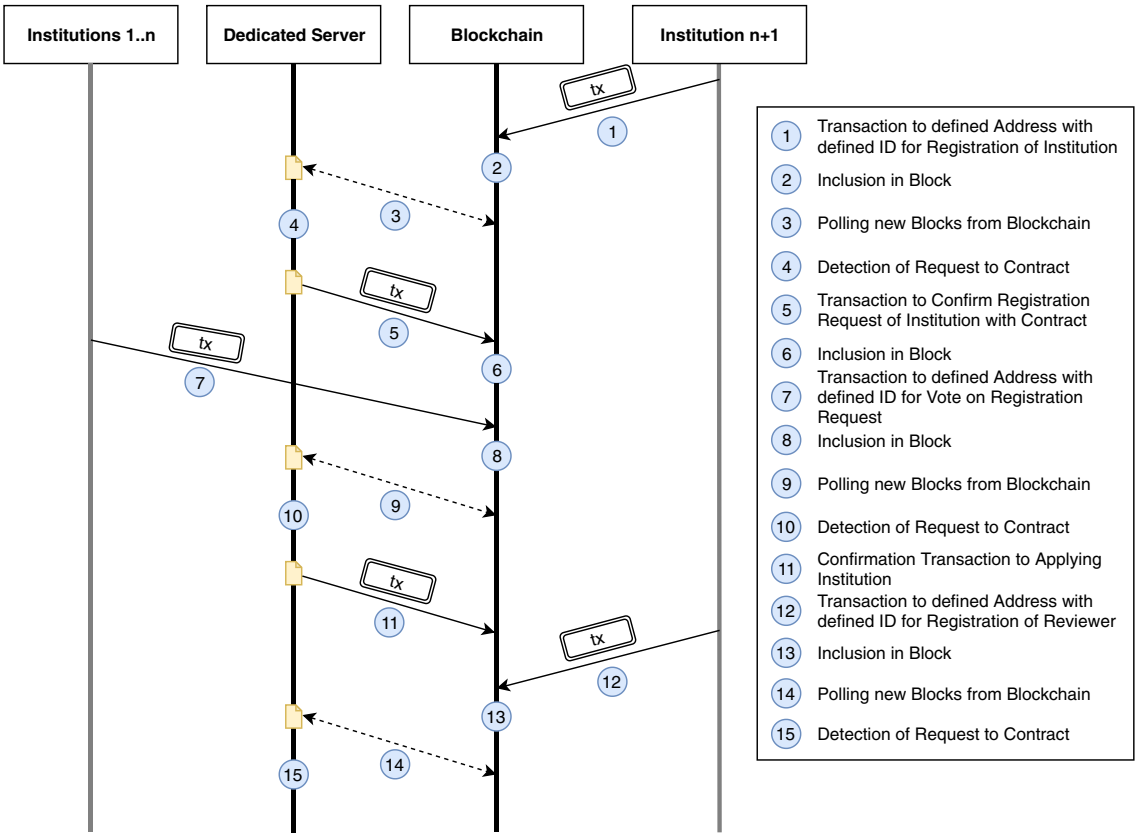

Fig. 6 Dynamic of the emulated institution list smart contract

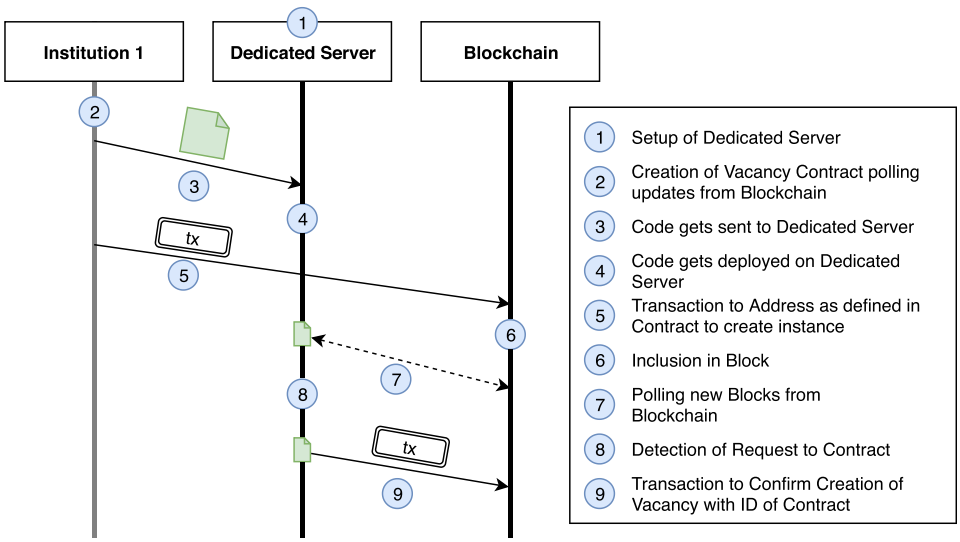

Fig. 7 Creation of the emulated vacancy smart contract

step and then the creation of the actual smart contract code which is sent and deployed on the server (steps 2–4). Following that, a vacancy instance can be created by issuing a transaction with the "create" keyword to the address specified in the contract (step 5). This will be detected by the contract code which polls the blockchain updates and then results in a confirmation transaction which contains the id of the newly created instance of the contract (steps 6–9).

The registration dynamic for vacancy contracts works the following (Fig. 8): An applicant sends a transaction with the keyword "register" and a previously specified quantity of currency to the address specified in the contract and the ID of the specific instance he wants to register with.

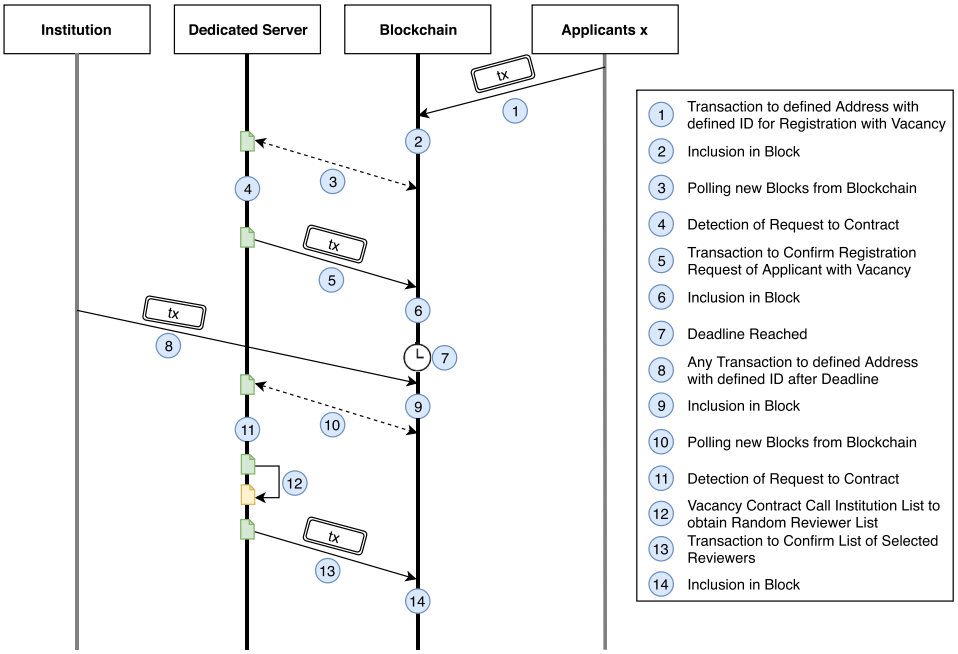

Fig. 8 Registration of the emulated vacancy smart contract

Following this, the transaction will be included in the blockchain (step 2), polled by the server (step 3) and detected (step 4). Then, similar to the institution list contract, the vacancy contract responds with a transaction to the user to confirm the registration (if successful) or with a refund if the deadline has been reached or not enough currency was attached (step 5-6). After the deadline has been reached (step 7) any transaction to the specified address with the correct ID will trigger the selection process of the reviewers (steps 8, 9). Then, the server polls and detects this transaction (steps 10, 11). The *Vacancy Contract* will directly invoke the *Institution List Contract* (from server to server without blockchain as an intermediary), including the random seed based on the block number (step 12). This read request will return the list of randomly selected reviewers which are then posted on the blockchain to confirm the selection in a transparent manner (steps 13, 14).

The dynamic of each application phase is depicted in Fig. 9. Similar to the native setup, the applicant stores its data which it encrypted with the public keys of the reviewers on an external server and then sends the hash of the actual data to the blockchain addressed to the address specified in the contract including the "hash" keyword. This is then included in the block, polled and detected by the contract which then confirms the reception in a transaction to the blockchain (steps 3–6).

Afterwards the reviewer gathers the information from the personal storage, decrypts the data with its private key and obtains the hash in a read operation from the dedicated server (steps 8, 9). Following that, the reviewer analyzes the data and sends the review to the blockchain in a transaction to the address of the smart contract (step 10), with the ID of the instance and the "review" keyword. This is then again, included in the block, polled and detected by the emulated smart contract (steps 11–13) which then confirms the reception of the review in step 14. After the deadline is then reached, the final result of the reviews is calculated from the reviews (step 16) and any applicant but also anyone else can issue a read request to obtain the results from the API of the server.

4 Prototype

To prove the viability of the proposed approaches we developed both proposed solutions.

4.1 Native smart contract

The native smart contract has been developed in the programming language *Solidity* [11] and deployed on the *Ethereum* testnet *Ropsten* [29]. This has been used in combination with Remix [28] to calculate the execution cost in the form of *Gas*[2]. To compare the resulting *Gas* cost with the real-world cost of this process we considered the median *Gas* price in Ethereum of the last three months which is 14.66*Gwei*. The current cost of participating in the selection processes in Brazil lies between approximately

[2] *Gas* represents the execution cost of an operation on the *Ethereum* blockchain. This cost of *Gas* is fixed for each type of operation depending on the resource usage.

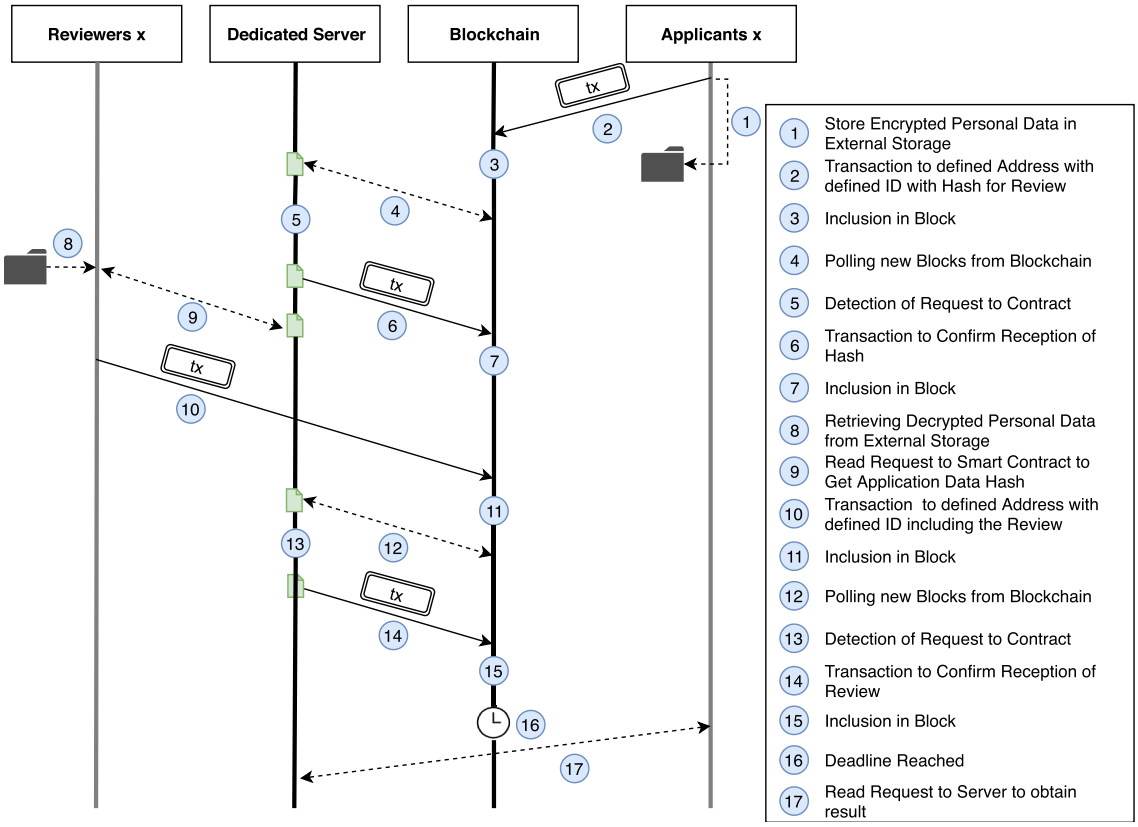

Fig. 9 Emulated vacancy smart contract application phase dynamic

66$ full time and 15.77 half time, per candidate[3]. Since the *Gas* price per operation is constant all operations have only been executed once.

4.2 Emulated smart contract

The emulated smart contracts have been developed in *javascript* using the *node.js* and *express* [14] frameworks. For data storage we used *MongoDB* [22] to store the data of each contract instance, allowing to run several instances of the smart contract in parallel.

We executed the scripts on a Dell Inspiron 5567 laptop with an i5-7200U quad-core 2.50GHz CPU and 8 GB RAM. For the blockchain we chose the *Steem* blockchain for the application due to the vast quantity of available documentation, easy setup and the fast execution times (3 s block production). The execution cost on the *Steem* blockchain is calculated in *Resource Credits* (*RC*) which are required to execute contracts on the blockchain. In order to obtain *RC*, currency has to be staked or staked currency has to be rented from another user. Per unit of currency (1 *STEEM POWER (SP)*)[4], approximately 1987676410 *RC* are available for the user. Since the RC cost per operation fluctuates with the network load, we executed all transactions 10 times and took the average.

All transactions have been executed on the public blockchain (not on the test-net). To obtain the required value of RC per operation we measured the available quantity before and after the execution of the transaction. Since *Resource Credits* recover around 10% per day, we executed the tests on an account with 100 STEEM Power. and added the minutely recovered quantity (of RC) to the measured execution cost to avoid the recovery rate influencing the result (Thus the resulting cost is slightly higher than the actual cost). All executed transactions can be seen on the block explorer[5]. We used two transaction types, namely there is the *"transfer"* type to transfer cryptocurrency between several accounts and the *"customJson"* type which allows to post *json* files on the blockchain. The smart contract posts responds to any correct invocation on the blockchain to allow verifying the correct execution of the single dedicated server for every step. Nonetheless, even

[3] All values in this context are given in US dollar and converted either from *Ethereum* with a conversion rate from 1:266 or from Brazilian Real with 1:0.26920.

[4] Where *Steem* is the name of the platform (blockchain) *STEEM* its cryptocurrency and *STEEM POWER (SP)* the staked form of *STEEM*.

[5] https://steemd.com for @hrm-user and @hrm-institution

Table 1 Unit conversion for execution cost calculation

Blockchain	Steem	Ethereum
Currency	STEEM	Ethereum
Execution Price Unit	RC	Gas
Currency to Ex. Price	1987676410 RC	68212824 Gas
Currency Price	0,45	360

without this extra data (resulting in an extra cost) it is possible to verify the correctness of the overall process if only the final result is included in the blockchain.

Due to the additional cost of the dedicated server we considered an additional fixed cost of 7$ for hosting the smart contracts on a dedicated server[6].

5 Evaluation

This section aims to compare the cost of the native implementation with the cost of the emulated smart contract. This includes the deployment cost, the maintenance cost and the cost per applicant. To account for the volatility of the currencies we used the max price of Ethereum and STEEM over the last three months (360$ of Ethereum and 0.45$ of STEEM). The conversion of the units can be seen in Table 1.

5.1 Deployment

The deployment cost of both solutions is depicted in Fig. 10. While the deployment of the *Institution List* using native smart contracts on the Ethereum blockchain costs 8.46$, using emulated smart contracts on the Steem blockchain it is 3.54$ of which 3.5$ are spent on the dedicated server. Since several contracts can be hosted on the same dedicated server, the 3.5$ stay constant with emulated smart contracts and only 0.04$ are necessary for each additional instance (additional server power is only required after a certain threshold of instances and read requests).

If we consider the deployments of both smart contracts, *Institution List* and *Vacancy*, the difference of the *Vacancy* contract is much bigger with native smart contracts due to its additional memory and computing overhead. While the price of the deployment using emulated smart contracts is basically the same (3.54$ of which 3.5$ are also due to the server hosting), using native smart contracts the price more than doubles if we compare the *Institution List to* Vacancy (8.46$ to 19.69$) per contract deployment.

[6] https://www.heroku.com/pricing

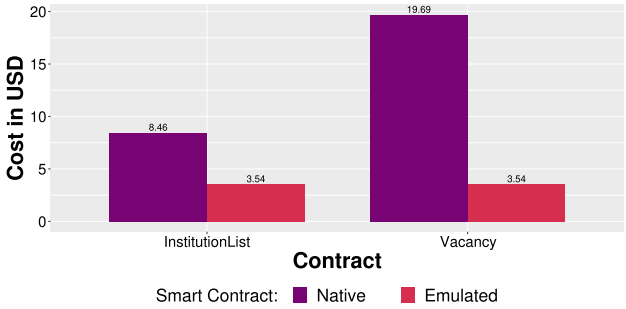

Fig. 10 Contract deployment cost

5.2 Maintainance

As visible in Fig. 11 the comparison in the context of the maintenance of the Institution List is much more complex. Since with emulated smart contracts the cost is only based on the number of bits in the *json* and transfer operations (which has to be included in the blockchain), operations using native smart contracts blockchain are also taxed in terms of memory occupation and computing overhead. Thus, using emulated smart contracts, any vote as well as any registration operation results in the same cost, while with native smart contracts (on *Ethereum*) "True" and "False" votes as well as the first vote come with a slightly different price. The same can be applied to the addition of reviewers where the overhead is the same using emulated smart contracts, using native smart contracts adding the first reviewer of an institution has a slightly different cost (due to the setup cost of the data structures).

Nonetheless, it is still very clearly visible that the deployment of the emulated smart contract has a cost advantage of up to 20 times in the case of addition of reviewers. The difference in the case of registration of a reviewer or the last deciding vote (finalize) are slightly smaller. This occurs since we opted to do registration via the transfer operation which comes with a higher base cost and thus also an overally higher cost.

5.3 Vacancies

The total cost of the selection process depends essentially on the number of reviewers the institution defines, the number of phases and the quantity of applicants.

Figure 12 shows the complete simulation of the contract with 25, 50 and 100 applicants while maintaining the number of reviewers at 5 and the number of phases at 4 (similar to the contract described in Sect. 2.3). While we understand that the values of the cheapest operations of the simulated smart contracts are difficult to read, we maintained the graph in the current form to display the strong cost differences between the two solutions.

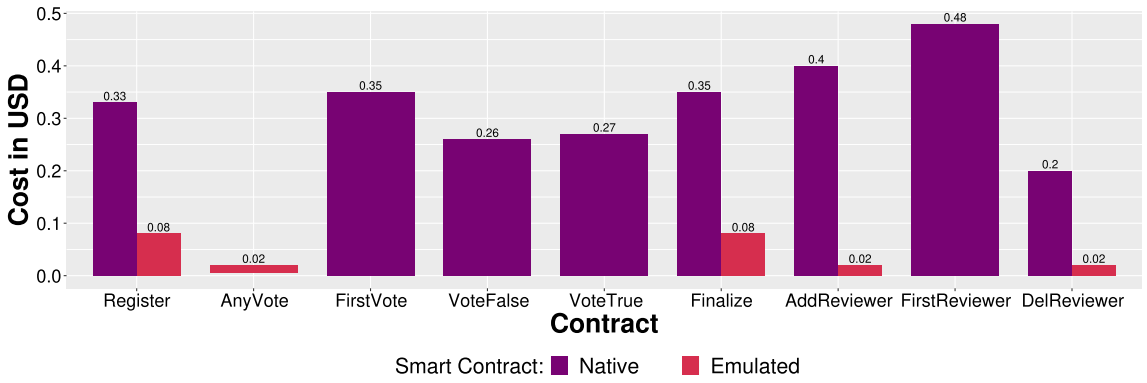

Fig. 11 Institution list maintenance cost

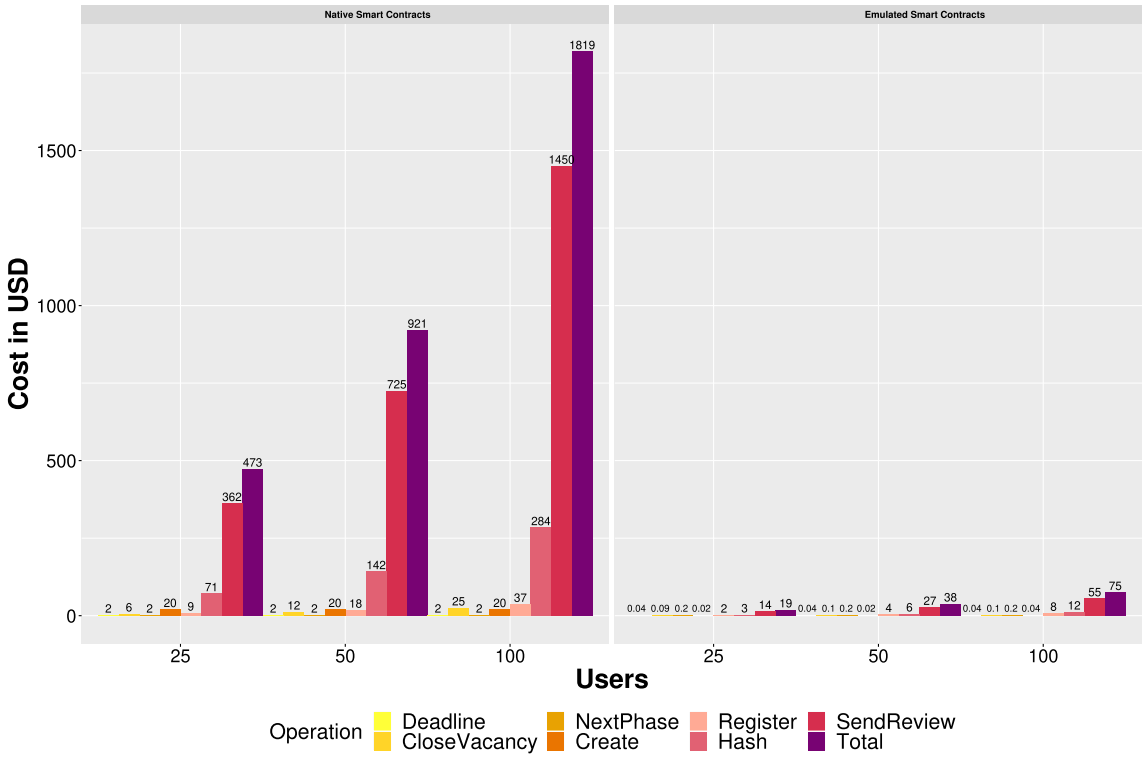

Fig. 12 Vacancy simulation cost

As depicted in the Figure, the main cost, are not computationally heavy tasks, but tasks which have to be executed numerous times (also heavily related to the transaction base costs). The most expensive operation (*"SendReview"*) has to be sent every phase, for every reviewer and every applicant (*phases* × *reviewers* × *applicants*). This is followed by the cost of sending the hash (each phase for each applicant) and the registration of the applicants (once per applicant). Following that there is the cost of the setup of the smart contract and the switching of the phases.

Both the native smart contracts as well as the emulated smart contracts follow similar tendencies in this case. Nonetheless, there is a significant difference in price. The execution of the native smart contract on *Ethereum* costs around 20 times more than the emulated smart contracts on *Steem*.

Native smart contracts, thus, result in a fixed cost of around 18$ per selection process and a per user cost of over 23$. The emulated smart contracts, in this case, have a fixed cost of around 0.3$ for the execution plus 7$ for the dedicated server. The per user cost of 0,75$ is also significantly under the cost of using native smart contracts. Additionally, we have to note that while on the *Ethereum* blockchain we pay to run this entire process only once, on *Steem*, since we obtain the Resource Credits by staking the

currency, we can execute one process every 10 days (due to the recovery rate of credits) while only having to pay once the cost of the server to run the scripts.

Considering the peak prices of *Ethereum* (the currency) and *STEEM* to run these processes, and considering that in the analyzed selection process each applicant has to pay between 15.77$ (emulated contract) and 66$ (native contract). This makes the approach using native smart contracts only viable for the more expensive processes and doesn't leave a lot of space for rewarding the reviewers and covering the costs of the process for the institutions. On the contrary, using emulated smart contracts there is plenty of space for rewarding the reviewers for their honest participating and paying the institution for the cost of running the application process.

Nonetheless, there are several security implications when running this process on the emulated smart contracts which will be discussed in the next section.

6 Discussion

This section will discuss concerns related to the proposed models and how it could be manipulated. In this context we also discuss the safety implications related to the differences between emulated and native smart contracts.

In terms of privacy concerns, the user data in our proposed model is more protected than in the classical selection process since in our model all the data is encrypted by the user using the reviewers public keys and stored at a place where the reviewers can access it only the reviewers themselves (and no administrative intermediates or people with access to the servers) have access to the user data. To guarantee that all reviewers receive the same data, the hash of the data is stored on the blockchain. This way, the reviewers can confirm the integrity and correctness of the data by comparing the hash of their encrypted data with the hash on the chain.

In relation to anonymity, all applicants and reviewers maintain pseudo-anonymity since they are only either identified by a chosen pseudonym or by their public key. The public key of the reviewer is only tied to the institution which is the only organ which knows the true identity of the reviewer enforcing a certain level of accountability. Nonetheless, to avoid corruption, the institutions don't have knowledge of the identities of the reviewers of the other institutions.

Similarly, the reviewers, during the selection process, find out the identities of the applicants through the submitted data. Thus, to increase the anonymity, the applicants should create a new key pair for each vacancy process they participate in. Since users are prone to avoid that (due to the additional work), they can be forced to create a new key pair through a dedicated front-end. In relation to the reviewers, it is possible to change the public key of the reviewers regularly to make it more difficult to tie them to real-world entities (Resulting in an increased maintenance cost).

Transparency is improved by pushing all the operations on the blockchain, in this sense, it is transparent how many people participate in a process, how many reviewers out of a pool of a certain size are chosen and, at the end of the process, which applicants passed which phases based on the reviews of which reviewers.

Additionally, due to the supplied deadlines in the contract, the transparency of the process is always guaranteed since results have to be delivered until the deadline. Unlike in the case of the real-world statutes where there is no stopping the institution from supplying the results after a given deadline or not supplying results at all.

While, in the case of emulated smart contracts, the institutions might not supply any results, or supply them after the deadline. The results, since the data is stored on the blockchain, can already be calculated. If an institution allows late registrations or reviews, users can go to court against the institution since this goes against the previously defined contract. Nonetheless, in the case of emulated contracts this is not naturally enforced by the blockchain.

In this case, the emulated and native smart contracts, have some fundamental differences. Native smart contracts are completely transparent since the code of the smart contract is published on the blockchain and thus publicly available. This way everyone the decentralized execution guarantees the deadlines and rules of the process.

Different to the native smart contracts, emulated smart contracts are only executed on a subset of servers or even on only a single server. Since these servers might be run by the institution which also created the contract or a single external third party, these servers might produce incorrect results or have their API return invalid data. This way, to obtain a similar level of transparency, the code has to be published externally. Nevertheless, there is no guarantee that the published code is exactly the same code as the one running on dedicated server. This can only be guaranteed by verifying all operations of the selection process and comparing the outcome with the published code. Since all results are on the blockchain transparently, this can be verified easily. An applicant, other institutions or in the case of public institutions any citizen could, for example, run their own instance of the emulated contract but disable the confirmations to the blockchain. This way anyone can compare their results of the process with the results the API of the centralized server returns. This allows to easily detect inconsistencies. In case of failure of servers, for some time, or completely since all results are on the

blockchain, a server can easily re-run the entire process to obtain the same state.

As discussed earlier, manipulating the process in our proposed system is more difficult compared to the current model since a set of unknown random reviewers of random institutions is selected which makes it much more unlikely to have a corrupt selection board.

Nevertheless, how safe the process actually is, depends on the defined parameters of the smart contract. With an increasing number of possible reviewers and an increasing number of selected reviewers the safety of the process improves gradually. For example: If an institution takes part in an institution list of four institutions, choosing a random reviewer of a certain area is easier to corrupt as if it is in a list of 100 institutions. Similar, if the vacancy only selects two random reviewers it will be not as secure against corruption as if it selects 10.

In the case of public institutions, how many reviewers are selected could be defined by law to guarantee an unbiased process. Since the smart contract code is public, everyone is able to verify if the contract obeys the given rules. In the case of emulated smart contracts, an external tool can be used to verify this. The development of such a tool can be simplified by creating a certain standard for public institutions which is followed by all institutions of a certain country. This way one tool can be used to verify the correctness of all the selection processes which follow this standard. Still, even without a specific tool, as mentioned earlier, anyone can execute the published code to verify the results. Thus, in order to guarantee the correctness of the process all the code of the emulated smart contracts has to be open source while native smart contracts are automatically open source since their code is deployed on the public ledger directly.

Another important topic of this process is the complexity inherent from the interaction with the blockchain, most users are not used to this environment and might face issues handling wallets, transactions and transaction data as well as cryptocurrencies. Additionally, we cannot expect users to handle encryption, decryption and verification of hashes. For this reason, most of this complexity should be hidden behind a user-friendly interface which allows the client to interact with the blockchain as it would be the homepage of the institution they are applying to. The main difference is that the blockchain is used as the underlying storage and execution environment which makes this process more transparent and secure since anyone is able to develop an interface which can display and send the required information.

Additional smart contracts can be developed to help to decentralize the elaboration of oral and written exams. Scans of the written or records of the oral exams can then be also sent to the reviewers and the hash stored on the blockchain.

While public institutions are obligated to notify the applicant about the results, in the industry, in practice, due to discrimination laws, companies often avoid notifying the candidate about the result. Our proposed model also considers this, as, for example, only the ranking of an applicant can be calculated in the smart contract. This way, while the users know when and in which step of the process they were disqualified they still do not know the reasons.

Finally, due to the volatility of the prices of *Gas* and the price of *Ethereum* other blockchains offering smart contracts (as EOS [15]) can be chosen. To further heavily decrease the cost and impact of volatility, emulated smart contracts can be used on a Blockchain as *Steem* since the main cost of the emulated contracts comes from the maintanance of the dedicated server (which is independent of the price of the cryptocurrency). Nonetheless, while these contracts can be made legally binding, emulated smart contracts do not have enforced deadlines on the blockchain layer which makes it easier to meddle with the process. Thus, tools have to be developed to detect these inconsistencies and a legal framework has to be available to punish the infringements.

Nonetheless, while in this case emulated smart contracts show decent advantages, this does not apply in all usage scenarios of smart contracts. In our case we have parties (the institutions) which are known and can be held accountable for their actions as well as applicants which will identify with the institutions and can be, a posteriori held accountable for their actions and, therefore, lose the advantage they gained from gaming the system. This does not apply to usage scenarios which involve transfers of assets between two not trusted pseudo-anonymous parties.

7 Related work

This section compares existing approaches from the literature and economy with our proposal. Smart contracts have been studied exhaustively already in the literature [34]. More specifically, the creation of groups on the blockchain with the help of smart contracts also found application in numerous studies. *Smart Pools*, for example, pool different miners together to decentralize mining pools and distribute mining rewards fairly [21]. *Decentralized Autonomous Organizations* (DAO) on the other hand have also risen more recently to give access to decentralized resources to a pool of people [25]. Usually in DAOs a group of people votes on proposals do distribute a pool of rewards to a common cause. The creation of these groups is fairly similar to our approach but voting and participating

Table 2 Comparison of approaches

	Curriculum validation	Transparency	Decentralization of evaluation	Privacy
Ouna	–	–	–	✔
Cverification	–	–	–	–
Disciplina	✔	–	–	✔
Hirematch	✔	–	✔	–
Appii	✔	–	–	–
Caerusconnections	–	✔	–	–
Our proposals	✔	✔	✔	✔

within these groups usually depends on the held stake of a certain asset. While in our case we build a more restricted model where the existing institutions in the list vote on the entry of a new member and base their decision on the real world entity of the registering institution. As described previously, each of the universities publishes the public key they use to identify on the blockchain within a certificate (on their website or similar for example) to harden this connection. Thus, differently to DAOs there is a sense of responsibility and, contrary to most DAOs, our proposed model is not subject to many of the issues pointed out in [9].

Projects related specifically to application processes are usually dedicated distributed ledgers to these kind of use cases.

One of these projects is *Ouna* [26]. *Ouna* allows candidates to define an anonymous profile based on a questionnaire. Companies fill in a similar questionnaire to get candidates based on this. The initial user profile does not contain personal data which protects the anonymity and privacy of the candidate. Then the candidate can exchange personal information only when he is willing to do it. Nevertheless, while this seems like a handy tool for companies it does not do the validation of the curriculum of the applicant.

Cverification, aims at facilitating the verification of curriculum elements [10]. Similar to *Ouna* it focuses on matching companies and possible candidates. For the verification of the diplomas, certificates and licenses, it will rely on the institutions which created the certificates to sign them on the blockchain to remove the need for a background check.

Similar to the previously discussed solutions, *Disciplina* focuses on matching companies and applicants [12]. They enhance this by cooperating with educational institutions which verify student curricula. This way they protect the employer from falsified diplomas.

Different to the previous solutions *Hirematch* offers the possibility to match companies and candidates and also allows companies to outsource the selection process to specific agents on the system which will be rewarded for the verification [16].

Also, there are other solutions ([3, 7]) but, in our point of view, they don't differ significantly from those already cited.

As shown in Table 2, none of the existing solutions protect the applicant and the hiring company at the same time. While some of them like *Hirematch* decentralize the evaluation they do not offer any level of privacy or transparency to the applicant. While other approaches try to improve the transparency and privacy issues, they rely on the educational institutions to verify the diplomas and lack a decentral validation method. Our proposed solution will validate the curricula and diplomas in a decentralized way while maintaining transparency and privacy of the user by not storing unencrypted data on the blockchain.

A survey in [4] compares different types of smart contracts on different blockchains but leaves out the possibility of emulated smart contract.

Thus, in regard to the comparison of emulated and native smart contracts, to the best of our knowledge, there are no academic works comparing the advantages and disadvantages in terms of safety and doing a thorough cost analysis.

8 Conclusion

We conclude that both our proposed systems allow institutions to automatize certain parts of the selections processes, and therefore, reduce the work of the participating committees. This way it not only adds additional levels of transparency but also of trust. Besides that, they also offer incentives to applicants and reviewers to meet the given deadlines and participate honestly in the system which further increases the probability of a fair and honest process. This way the trust in public selection processes can be improved and a fair and unbiased process for all applicants can be guaranteed more easily.

The emulated smart contracts, had a significant cost advantage compared to native smart contracts. Additionally, we conclude that this advantage would also hold if emulated contracts were executed on other popular blockchains as Thron, bitshares or EOS [15, 30] because the

main cost of the emulated approach is the dedicated server which is not subject to fluctuations of the price of the cryptocurrency.

Thus, we recommend the use of emulated smart contracts when the main concern of the usage scenario is to provide transparency and immutability to detect possible inconsistencies in certain processes and the participating parties can be held accountable after the detection of unfair behaviour after the execution of the process.

In the future we want to study different smart contract usage scenarios where, in the literature, native smart contracts are used, but emulated smart contracts could also be applied to reduce the cost of the execution (for example in the field of IOT or in the field of autonomous vehicles).

References

1. Al-Bassam, M., Sonnino, A., Bano, S., Hrycyszyn, D., Danezis, G.: Chainspace: A sharded smart contracts platform. arXiv:1708.03778 (2017)
2. Angraal, S., Krumholz, H.M., Schulz, W.L.: Blockchain technology: applications in health care. Circulation **10**(9), e003800 (2017)
3. Appii: Employee background checks and cv verification underpinned by blockchain technology. register now!. https://appii.io/ (2018). Accessed 22 Aug 2019
4. Bartoletti, M., Pompianu, L.: An empirical analysis of smart contracts: platforms, applications, and design patterns. In: Proceedings of the International conference on financial cryptography and data security, pp. 494–509. Springer (2017)
5. Becker, B., Gerhart, B.: The impact of human resource management on organizational performance: progress and prospects. Acad. Manag. J. **39**(4), 779–801 (1996). https://doi.org/10.5465/256712
6. Cachin, C.: Architecture of the hyperledger blockchain fabric. In: Proceedings of the Workshop on Distributed Cryptocurrencies and Consensus Ledgers, vol. 310 (2016)
7. Caerusconnections: Find a career you'll love today. https://www.caerusconnections.io/ (2018). Accessed 22 Aug 2019
8. Catallini, C.: How blockchain applications will move beyond finance. Harvard Business Rev **2**, (2017)
9. Chohan, U.W.: The decentralized autonomous organization and governance issues. SSRN 3082055, (2017)
10. Cverification: Blockchain-based recruitment and background verification platform. https://cverification.com/ (2018). Accessed 22 Aug 2019
11. Dannen, C.: Introducing Ethereum and Solidity. Springer, New York (2017)
12. Disciplina: Disciplina -we are developing the first blockchain to create verified personal profiles based on academic and professional achievements. https://disciplina.io/ (2018). Accessed 22 Aug 2019
13. Etherscan.io: Ethereum gas price history. https://etherscan.io/chart/gasprice (2019). Accessed 22 Aug 2019
14. Express: Fast, unopinionated, minimalist web framework for node.js. https://expressjs.com/ (2019). Accessed 22 Aug 2019
15. Grigg, I.: Eos: an introduction. http://www.org/papers/EOS_An_Introduction.pdf (2017). Accessed 22 Aug 2019
16. Hirematch: Hirematch connects job seekers and job finders using the blockchain and the cryptocurrency 'hire'. https://hirematch.io/ (2018). Accessed 22 Aug 2019
17. Huselid, M.A.: The impact of human resource management practices on turnover, productivity, and corporate financial performance. Acad. Manag. J. **38**, 635–672 (1995). https://doi.org/10.5465/256741
18. MARTINS, C.: Mp deflagra operação contra fraudes em concursos públicos de seis municípios do rs. https://gauchazh.clicrbs.com.br (2018). Accessed 22 Aug 2019
19. MARTINS, C.: Mp denuncia nove pessoas por fraude em concursos públicos de prefeituras. https://gauchazh.clicrbs.com.br (2018). Accessed 22 Aug 2019
20. Martins, V.: Mp-go denuncia 26 pessoas por envolvimento em fraude em concurso para delegado em goiás. https://g1.globo.com (2017). Accessed 22 Aug 2019
21. McCorry, P., Hicks, A., Meiklejohn, S.: Smart contracts for bribing miners. In: Proceedings of the International Conference on Financial Cryptography and Data Security, pp. 3–18. Springer (2018)
22. MongoDB: The database for modern applications. https://www.mongodb.com/ (2019). Accessed 22 Aug 2019
23. Nakamoto, S.: Bitcoin: A peer-to-peer electronic cash system (2008)
24. Neiheiser, R., Inácio, G., Rech, L., Fraga, J.: Hrm smart contracts on the blockchain. ISCC '19. IEEE (2019)
25. Norta, A.: Creation of smart-contracting collaborations for decentralized autonomous organizations. In: Matulevičius, R., Dumas, M. (eds.) Perspectives in Business Informatics Research, pp. 3–17. Springer, Cham (2015)
26. OUNA: Find your dream employer. https://ouna.io/ (2018). Accessed 22 Aug 2019
27. PB, G.: Saiba quais são os 98 concursos que teriam sido fraudados por investigados na operação gabarito. https://g1.globo.com (2017). Accessed 22 Aug 2019
28. Remix: Solidity compiler. https://remix.ethereum.org/ (2019). Accessed 22 Aug 2019
29. Ropsten: Ropsten testnet. https://ropsten.etherscan.io (2019). Accessed 22 Aug 2019
30. Schuh, F., Larimer, D.: Bitshares 2.0: General overview (2017)
31. Sciascia, D., Pedone, F., Junqueira, F.: Scalable deferred update replication. In: Proceedings of the 2012 42Nd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN), DSN '12, pp. 1–12. IEEE Computer Society, Washington, DC, USA (2012)
32. StateoftheDApps: State of the dapps - ranking the best ethereum, eos & steem dapps. https://www.stateofthedapps.com/rankings (2019). Accessed 22 Aug 2019
33. Szabo, N.: Formalizing and securing relationships on public networks. First Monday (1997)
34. Tariq, N., Asim, M., Al-Obeidat, F., Zubair Farooqi, M., Baker, T., Hammoudeh, M., Ghafir, I.: The security of big data in fog-enabled iot applications including blockchain: a survey. Sensors **19**(8), 1788 (2019)
35. Vukolić, M.: The quest for scalable blockchain fabric: proof-of-work vs. bft replication. In: Camenisch, J., Kesdoğan, D. (eds.) Open Problems in Network Security, pp. 112–125. Springer, Cham (2016)
36. Wood, G.: Ethereum: a secure decentralised generalised transaction ledger. Ethereum Project Yellow Paper **151**, 1–32 (2014)
37. Zhao, J.L., Fan, S., Yan, J.: Overview of business innovations and research opportunities in blockchain and introduction to the special issue. Financ. Innov. **2**(1), 28 (2016). https://doi.org/10.1186/s40854-016-0049-2

Ray Neiheiser Did his B.S. at the University of Media in Stuttgart and since then has earned his received degree in Computer Science from the Federal University of Santa Catarina (UFSC) in 2017. At the moment he is a Ph.D. student at the same university. His main areas of interest are the construction of reliable distributed systems and electronic governance. In the context of this, he specifically focuses on byzantine fault tolerant consensus and the distributed ledger technology.

Gustavo Inácio Is a undergraduate student in the field of computer science at the Federal University of Santa Catarina. He participated in this work in the context of a scientific internship in the distributed systems laboratory. His main interests include Cloud Computing, Security, Distributed Systems, and Games.

Luciana Rech Is an Associate Professor at the Informatics and Statistics Department (INE)of the Federal University of Santa Catarina(UFSC) and a member of the Distributed Systems Research Laboratory (LAPESD). Graduated from University of Cruz Alta in Computer Science, with Master's degree in Computer Science (Field: Parallel and Distributed Computing) from Federal University of Santa Catarina and Ph.D. in Electrical Engineering (Field: DAS/Information System). She has experience in the field of computer science with a focus on computational systems working more closely with: Distributed Systems, Intelligent Systems, Real Time Systems and Applied Informatics.

Joni Fraga Received the B.S. degree in Electrical Engineering in 1975 from the University of Rio Grande do Sul (UFRGS), the MSE degree in Electrical Engineering in 1979 from the University of Santa Catarina (UFSC), and the Ph.D. degree in Computing Science (Docteur de l'INPT/LAAS) from the Institut National Polytechnique de Toulouse / Laboratoire d'Automatique et d'Analyse des Systèmes, France, in 1985. Also, he was a visiting researcher at UCI (University of California, Irvine) in 1992–1993. Since 1977 he has been employed as a Research Associate and later as a Professor in the Department of Automation and Systems at UFSC, in Brazil. His research interests are centered on Distributed Systems, Security and Fault Tolerance. He has over 280 scientific publications and is a Member of the IEEE and of Brazilian scientific societies.