Blockchain Ecosystem for Verifiable Qualifications

Diogo Serranito
Instituto Superior Técnico, Universidade de Lisboa
Lisbon, Portugal
diogo.serranito@tecnico.ulisboa.pt

***Abstract*—Human resource contracting processes depend on trustworthy qualifications (diplomas or certificates) that may be counterfeit or falsified and are hard to verify manually. This paper leverages blockchain technology and smart contracts, to enforce a decentralized verification solution for higher education certificates, e.g., university diplomas. The solution allows Higher-Education Institutions (HEI) to register the certificates they issue in the blockchain, and recruiting organizations to check the authenticity and integrity of these certificates. The solution is implemented through five major components: i) the consortium smart contract, ii) the HEI smart contract, iii) the HEI client, iv) the recruiter app, and v) the consortium app. A prototype of the implementation is tested in a real blockchain (a testnet) and the challenges raised with the experiment are identified and discussed with a main focus on the decentralization mechanism performance.**

***Index Terms*—Authenticity, Blockchain, Document verification, Education certificate, Ethereum, Integrity, Smart contract.**

I. Introduction

Recruitment of qualified personnel for an organization can be a lengthy process, as it may require analyzing a large number of candidate curricula. Moreover, *verifying* paper or digitized certificates can be a very time consuming process, as it may involve manually contacting the academic institution that granted them, which is necessarily costly in terms of time and resources consumed. However, *verification is necessary*, as there is a market of counterfeit academic diplomas and certificates. According to a recent survey, over half of the curricula and job applications (53%) contain falsifications and over three quarters (78%) are misleading [1]. Even politicians and other public figures are often found at the center of controversies over the qualifications they report.

Most academic certificates today are still issued in paper, then sometimes digitized to PDF or a similar format. *Digital signatures* based on public-key cryptography [2] can be used to provide the same or more security than the paper-based solution. However, digital signatures have to be verified with a public key, that must be bound to an identity and distributed in a secure way, i.e., its authenticity and integrity must be guaranteed. This problem is known and is solved with a Public-Key Infrastructure (PKI), that must include a set of organizations designated Certificate Authorities (CAs) [3]. CAs issue digital certificates (not to be confused with academic certificates) that contain bindings between a public key and an identity. This is a heavy solution with several difficulties, one of them the centralized nature of CAs.

A globalized world requires a solution for this problem that goes beyond *ad hoc* agreements or national-level structures. Moreover, the solution has to be *decentralized*, in the sense that it should not rely on third parties to support it, as no third party can be well-accepted in every country. It also needs to assure certain *cybersecurity* attributes, specifically authenticity, integrity and availability of the relevant qualifications data. It has to be *modular* to support expansions, and *scalable* to support an increasing number of players. *Blockchain* technology offers solutions to these challenges by providing decentralized, immutable (append-only), transparent, and trustworthy storing of data [4]–[7]. Furthermore, it allows organizations to remove intermediaries, reducing the time of going through third-parties, while also lowering the cost of transactions.

We propose such a *solution for the qualification verification problem*, i.e., for the verification of digital documents containing qualifications that we simply designate as *certificates*. To satisfy all the requirements, more than a system we need an *ecosystem*, that is flexible and can grow with the demand. Therefore, our solution is indeed an ecosystem based on a permissionless, open, blockchain. The solution allows Higher-Education Institutions (HEIs) to register the certificates they issue (e.g., graduation diplomas) in the blockchain (but not *storing* the certificates in the blockchain), and recruiting organizations to check the authenticity and integrity of these certificates. This ecosystem of HEIs and other organizations is being developed in the context of project *QualiChain*, that will assess it in the context of a set of pilots and add advanced features that are out of the scope of this article (e.g., career counselling, intelligent profiling, and competency evaluation) [8].[1] In fact, the ongoing digital transformation of the area creates many challenges [9]. A prototype of our solution is available online.[2]

At the current state of the art, and of the practice, the major software engineering challenge of such an ecosystem is arguably to ensure *decentralization*. Despite decentralization being one of the main selling points of blockchain and Distributed Ledger Technology, in practice many solutions end up being centralized, or as good as if they were centralized. In the paper we discuss two important dimensions of decentralization and show how our solution achieves them: *governance*

[1] https://qualichain-project.eu/
[2] https://github.com/QualiChain/consortium

978-1-7281-7091-6/20/$31.00 ©2020 IEEE

decentralization and *access decentralization*. There are others like *geographical* and *mining decentralization* [10].

The document is organized as follow. Section II discusses related work. Section III presents the background concepts, and the main solutions for qualifications. Section IV describes the ecosystem of the problem to be solved, and details the proposed implementation. Section V evaluates the obtained results. Section VI identifies the challenges that are posed to the software engineering field. Finally, Section VII concludes the paper.

II. Related Work

To the best of our knowledge *no similar ecosystem is available today*. There are a few preliminary proposals of systems for storing certificate data in a blockchain, but they are focused on a limited environment, not in providing a decentralized, scalable ecosystem that supports a consortium of HEIs.

Two recent projects addressing the problem of reliability of qualifications are Open Badges and Blockcerts.

Open Badges [11] are verifiable, portable, digital badges with embedded metadata about skills and achievements. Open Badges, from version 1.1 upwards, are Linked Data objects represented in JSON-LD format, defined under the relevant JSON-LD context (currently, Open Badges v2.0). They can be incorporated in digital images in PNG or SVG format that represent the certificate in a visual way, *e.g.*, with the information of the certificate holder and a signature and a stamp of the issuer (called badge baking), or be used as a stand-alone data packet. Open Badges can be stored online (preferable in data storage with Linked Data support) or offline as files, in which form they can also be easily transferred.

Blockcerts [12] consists of open-source libraries, tools, and mobile apps enabling a decentralised, standards-based, recipient-centric ecosystem, enabling trustless verification through Blockchain technology. The initial design was based on prototypes developed at the MIT Media Lab and by Learning Machine Technologies. An application example for MIT graduates: students receive their diploma through an app called Blockcerts Wallet; this tamper-proof diploma can then be verified by any employer or school [13]. Blockcerts uses Open Badges as certificates and Blockchain addresses as recipient identification. For simplicity of usage and security, the Blockcerts Wallet generates its own key pair, sending the respective public key to MIT, which stores it in a digital record. The student's certificate is generated as an Open Badge and the digital hash of the badge is stored on a public Blockchain (at the moment Bitcoin or Ethereum) and added to the badge, together with the id of the transaction that contains the hash on the selected Blockchain. This way, it is possible to verify the issuing date of each badge by looking at the date of the Blockchain block that contains the badge hash. Then, an employer can efficiently verify if a digital diploma is legitimate, through a portal where the diploma can be uploaded, and its hash compared with the one stored in the Blockchain.

There is other related work. Mikroyannidis et al. have studied the problem with a focus on the learning process [14]. The European Union under the guidance of CEF Digital is implementing the European Blockchain Services Infrastructure (EBSI), currently in version 1 [15]. One of the EBSI's initial use cases aims to support digital diplomas, but this is still in development.

Our ecosystem may be considered a Decentralized Autonomous Organization (DAO), but unlike most DAOs we do not aim to provide or use a cryptocurrency [16], [17]; on the contrary, our DAO is an ecosystem of HEIs that provides services to recruiters and alumni.

Despite all these examples, no ecosystem for granting diplomas is available today.

III. Blockchain and Smart Contracts

In simple terms, a *blockchain* is a decentralized computing system and a replicated trustworthy data store that maintains an immutable (i.e., append-only) data structure composed of a sequence of blocks [5], [6]. Originally, the term blockchain designated a component of the Bitcoin system [4], but today the term is used generically. Blockchains can be *permissionless* (anyone can join) or *permissioned* (permission to join is granted by an organization or a consortium of organizations). From this explanation, it becomes clear that permissioned blockchains suffer from governance centralization, so we opted for a *permissionless* blockchain.

Ethereum, the blockchain technology we target in the prototype, is one of the most popular blockchains. Ethereum is an open-source, public, distributed platform [18].[3] As a permissionless blockchain, it allows organizations to easily verify data stored there, e.g., certificates. Similarly to Bitcoin, Ethereum implements a cryptocurrency, denominated *ether*. However, it also supports the execution of *smart contracts* [18], [19] in the nodes, i.e., it is programmable. Essentially thousands of nodes contain and execute the same smart contracts. A smart contract can be considered to be an *object* in the object-oriented paradigm, as it has state stored in a set of local variables/attributes, a set of functions that allow changing that state, and the ability to invoke functions in other smart contracts. A client or a smart contract calls a function in a smart contract by issuing a transaction, that is executed by every node when the block that contains that transaction is appended to the blockchain. All transactions are authenticated by their issuer with a digital signature obtained using its private cryptographic key, which can be verified using the corresponding public key.

IV. The (Eco)System

An ecosystem is composed of one or more *consortiums*. Each consortium includes a set of HEIs. For instance, project QualiChain is bootstrapping a consortium with a few HEIs,

[3]Notice that our ecosystem is not limited to run in the Ethereum public network as there are several blockchains and technologies that are compatible with Ethereum smart contracts. However, we mention the blockchain as Ethereum in the paper for simplicity.

but eventually more HEIs will join and other independent consortiums may appear.

An ecosystem with a single consortium and three HEIs is represented in Figure 1. Data about the consortium is stored in the blockchain in a *consortium smart contract*. This smart contract contains data about the HEIs that are part of the consortium and the rules of the consortium (e.g., how a HEI can join). Each HEI is itself represented by a *HEI smart contract* that contains data about the certificates it has issued. All these smart contracts exist in a blockchain, e.g., Ethereum. HEIs run a *HEI client*, an application, that registers certificates in the HEI smart contract. The smart contract does not store the files of the certificates (PDF files), so they may be stored by the HEIs and the alumni, or in a decentralized file storage system like IPFS [20], Ethereum SWARM [21] or Metadisk [22]. Finally, recruiters run a *recruiter app* that allows them to verify if a certificate file was indeed emitted by a certain HEI.

The architecture in the figure starts illuminating what we mean by governance decentralization: there can be several consortiums and each one is managed by its HEIs. It also shows what we mean by access decentralization: there are no access intermediaries, as the blockchain is replicated in many nodes and all components are managed by their owners, who may even have several instances of them (e.g., a HEI can have several HEI clients).

All entities that compose the ecosystem interact with the Ethereum blockchain through a *JavaScript* application. Three different modules were developed and adapted to the needs of each class of entities:

- ***HEI Client:*** this module offers the possibility for registration and revocation of education certificates (Section IV-C).
- ***Recruiting App:*** this module allows recruiters to easily verify a candidate's education certificate (Section IV-D).
- ***Consortium App:*** this module allows new HEIs to join the platform through a consortium-based voting system (Section IV-E).

Next we present in more detail the different components of the ecosystem, starting with the two smart contracts:

- ***Consortium Smart Contract:*** accessed by the HEIs to manage the membership of the consortium and by the recruiters to get the address of the HEI's smart contracts (Section IV-A).
- ***HEI Smart Contract:*** accessed by all entities to store or get certificate hashes (Section IV-B).

A. Consortium Smart Contract

The consortium level is mainly the *consortium smart contract*. Without compromising decentralization, in our solution we consider that there is an entity that jumpstarts a consortium by deploying that contract into the blockchain. In our case, contracts were written in Solidity [23], the most adopted language for Ethereum. Before deployment, Solidity code has to be compiled to Ethereum virtual machine (EVM) code, which is the language actually executed by Ethereum nodes.

The consortium smart contract defines a mapping (a key-value data structure) named *contracts* that associates the identifier of every registered HEI to its smart contract address. For HEI identification we use *Decentralized Identifiers* (DIDs), as defined by the W3C [24]. These identifiers are designed to allow the entity identified by the DID, e.g., a HEI in our case, to prove control over the DID, and to be implemented independently of a centralized registry, identity provider, or certificate authority (CA). DIDs are essentially URLs that relate a subject to means for trustable interactions with that subject.

Here governance decentralization comes again into play: it is up to the consortium to decide if the HEI is allowed to join or leave, similarly to what happens in many organizations, such as private associations. When a HEI wants to join the developed platform, it deploys its smart contract (see next section) and makes a request to join the current consortium. This call is a manifestation of interest in joining that does not grant that effect automatically. In fact, other members of the consortium have to show their agreement. There are other options, but we designed a simple voting scheme. The basic functionality of the smart contract is then provided by three functions/operations:

- *registerHEI(id, address)*: vote to register the address of the HEI smart contract of a new HEI joining the consortium (recall that all function calls are authenticated using a signature, in this case the HEI's);
- *removeHEI(id)*: vote to delete a HEI's smart contract to cancel its registration in the consortium;
- *getHEI(id)*: return the HEI's contract address corresponding to the received identifier.

The registration of the HEI takes effect when the number of votes reaches a certain threshold T_v, that is also stored in a variable of the consortium smart contract.

A similar scheme is used for a HEI to leave the consortium. A number T_v of HEIs may call *removeHEI(id)* and force a HEI to be removed, e.g., due to some kind of misconduct.

The threshold value can be adjusted according to the consortium needs. Again this involves a voting scheme and two functions:

- *changeThreshold()*: vote to replace the current threshold with a new T_v value.
- *getThreshold()*: returns current value of T_v.

This voting scheme is another vector of governance decentralization, as it is up to the members of a consortium to manage different aspects such as who can join and how many votes are needed to take decisions. On the contrary, there is no centralized entity to manage and supervise the actions being performed. The voting scheme can be made more complicated and may mimic governance mechanisms used in private associations.

B. HEI Smart Contract

Similarly to what happens with the smart contract of the previous section, the *HEI smart contract* can have different

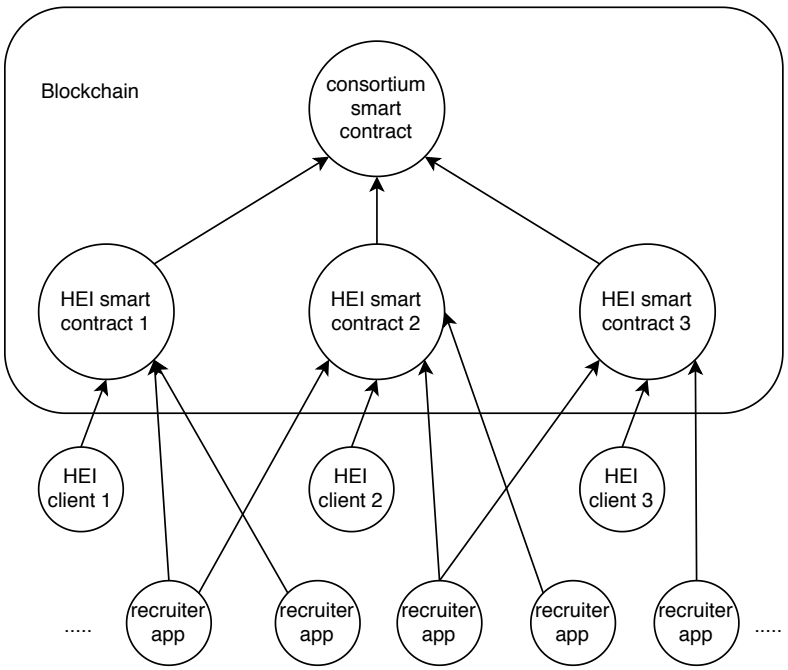

Fig. 1. Data flow diagram representation of an ecosystem with a single consortium of three HEIs. Each HEI may also run a Consortium App to manage the membership of the consortium (not represented).

versions depending on different design options, but we developed a single one, that we now present.

The HEI smart contract holds a mapping, *certificates*, that associates a *job seeker id* with the *cryptographic hash* of his certificate. There are many options for this job seeker id, e.g., it may be the civil id, the student id he had in the HEI, a DID, etc. Cryptographic hash functions, e.g., SHA-2 and SHA-3, have two important properties [25]:

- *one-way*: it is not possible to obtain the input from the output (the hash);
- *strong collision resistance*: it is computationally infeasible to find two different inputs that have the same hash.

In our case these properties mean that it is not possible to retrieve a certificate from its hash (one-way) and it is not possible to falsify certificates by finding two different certificates with the same hash (strong collision resistance).

The contract provides the following functions:

- *registerCertificate(id, hash)*: register a certificate on the blockchain by storing the job seeker id and a cryptographic hash of the certificate file; notice that the certificate itself is not stored in the blockchain, only its hash that has the property of being one-way;
- *revokeCertificate(id)*: revoke a certificate (this should be rarely used but may be necessary as mistakes can happen);
- *verifyCertificate(id)*: allows verifying if a specific certificate is registered and not revoked, by returning the hash or an exception.

The first two functions check if they were called by the HEI that owns the contract, and return an exception otherwise. The third can be called by anyone. This solution supports a single certificate per student per HEI but can be trivially extended to support more, e.g., by including a type of certificate in the *id*.

We present the Solidity code of the HEI smart contract in Figure 2. The security of smart contracts is an important concern today [26], [27], but this smart contract is small enough for us to argue it has no vulnerabilities, at the current state of knowledge about Solidity / Ethereum security. The consortium smart contract is larger, approximately 240 lines of code, but still very small when compared with common source code. Specifically, these sizes are small when be compared for example with PHP web applications that can have tens or thousands of lines of code [28], or with databases like MariaDB and PostgreSQL that have around 2 million each.[4] The number of vulnerabilities and bugs in general is known to depend strongly on the number of lines of code [29]. Moreover, this code can be checked using a code verification tool [30].

C. HEI Client

HEIs typically run an Academic Management System (AMS) such as Moodle, Blackboard or FenixEdu. When a student graduates, HEI staff interacts with the AMS to issue the student's certificate. The AMS will then generate the certificate and make it available for the staff, that typically prints it.

We designed a *HEI client* that automatically registers the certificate in the blockchain instead of printing it. We designed an interface with FenixEdu[5] (but it works identically

[4]https://github.com/AlDanial/cloc

[5]https://fenixedu.org/

```
pragma solidity >=0.4.22 <0.6.0;

contract HEI {
    address owner;
    mapping(uint => bytes32) certificates;

    constructor() public{
        owner = msg.sender;
    }

    modifier isOwner() {
        require(owner == msg.sender);
        _;
    }

    function registerCertificate(uint id, bytes32 hash) isOwner public{
        certificates[id] = hash;
    }

    function revokeCertificate(uint id) isOwner public{
        certificates[id] = 0;
    }

    function verifyCertificate(uint id) public view returns (bytes32 hash) {
        if(certificates[id] != 0) {
            return certificates[id];
        }
    }
}
```

Fig. 2. HEIs smart contract Solidity code.

with others) that monitors a folder, that we designate *registered_certificates*, where certificates are temporarily stored in PDF format. Whenever a new education certificate is stored in the *registered_certificates* folder, a watcher throws an event that leads to a call to the *registerCertificate(id, hash)* function of the HEI smart contract, where *id* is the job seeker id (taken from the filename or file metadata) and *hash* is the cryptographic hash of the PDF file. For that purpose, an Ethereum transaction is created. After the transaction is confirmed, the file is stored in a decentralized file system. In our prototype we store the file in IPFS by contacting an IPFS node [20]. IPFS returns a multihash (a self-describing hash) that can be used by anyone connected to the IPFS network to retrieve this file.

When there is the need to revoke a certificate registered on the blockchain, the HEI interacts with its AMS, that adds the certificate to be revoked to a shared folder named *revoked_certificates*. The HEI client has a second watcher that monitors that folder, leading to a call to function *revokeCertificate(id)* of the smart contract.

D. Recruiter App

When a recruiting organization opens a job position, there is the need to make a selection of the most appropriate candidates. Before or during this selection, certificates of the candidates need to be verified. For employers to receive a candidate's certificate, two approaches are possible: receive the certificate file from the candidate; receive the IPFS multihash corresponding and retrieve the certificate file from IPFS.

After acquiring the file with the certificate, the recruiter can use the *recruiter app* to verify the certificate authenticity/integrity. The recruiter app requests the job seeker id, the DID of the HEI, and the certificate file. Then the app calls the *getHEI()* function on the consortium smart contract and receives a hash that is compares with the hash of the certificate file. If they are identical, the file is authentic, otherwise it is not.

E. Consortium App

The process of accepting/dismissing HEIs from a consortium needs to be dynamic and reliable. This system was developed for members to have complete trust in it, therefore, transparency is another major property that needs to be assured. Blockchain technology provides this property, being one of the main benefits of adopting such a disruptive system. The *Consortium smart contract* is the core component of the voting system, where all polls and the respective votes are registered. Therefore, there is a complete transparency of the voting process, since the votes of all HEIs are openly available on the blockchain.

An interface was also developed to support the *Consortium App*, that will facilitate the interaction of members with the consortium contract. This interface provides three forms and lists of ongoing polls, one for each of the contract's main voting operations.

1) Register HEI: As stated previously, whenever a HEI decides to join the platform, it deploys its own *HEI smart contract* on the Ethereum blockchain and communicates its intents to the consortium by invoking the *registerHEI()* function.

When a consortium member runs its own *Consortium App*, it will start by querying the consortium contract for currently ongoing polls. This is accomplished by calling the *getPollingInfo()* function, that will return four different elements stored in the contract:

- *registerIdArray*: array containing the identifiers associated with the HEIs currently being voted to join the consortium.
- *registerContractArray*: array containing the HEI contract addresses associated with the HEIs currently being voted to join the consortium.
- *cancelIdArray*: array containing the identifiers associated with the HEIs currently being voted to leave the consortium.
- *thresholdVotingValue*: new value currently being voted to replace the current threshold.

The *Consortium App* interface will then be updated with the received information, that consortium members can interact with to vote in any of the currently available polls.

After receiving this information, consortium members can vote on this poll by interacting with the *Register HEI* form in the Consortium App interface. The *Consortium App* will then create a new transaction and sign it with the private key of the HEI that is using the application. This transaction will invoke the *voteRegisterHEI()* function in the *Consortium smart contract*, with the respective values as arguments.

2) Remove HEI: Even though a HEI leaving the consortium will be a rare occurrence, this may happen due to a variety of reasons. It can be the HEI itself that decides to leave or the other members agreeing on its exclusion. If this is the case, one of the members (including the HEI itself) can start a new poll by interacting with the *Remove HEI* form available on the *Consortium App* interface.

The respective poll will then be displayed in the interface by calling the *getPollingInfo()* on startup. When submitted, a transaction is created and signed with the HEI's private key, and sent to the blockchain network. This transaction invokes the *voteRemoveHEI()* function in the consortium contract, with the HEI identifier as an argument.

3) Change Threshold: For a poll to be successful/rejected, the number of positive/negative votes needs to reach the threshold value defined in the consortium contract. However, this value needs to be dynamic to match the fluctuation in the number of consortium members. Therefore, the *Consortium App* interface provides the *Change Threshold* form, which receives the new value to be proposed and starts a new poll to replace the threshold.

After having filled and submitted the *New value* field, the application will create a new transaction signed with the private key of the HEI utilizing the interface. This transaction is sent to the blockchain network and will be processed by executing the *changeThreshold()* function of the consortium contract. The function receives the new threshold.

Other members of the consortium can then vote if they accept or reject the alteration on the threshold value by initializing the *Consortium App*. On startup, the interface will be updated through the *getPollingInfo()*, providing the possibility to vote by interacting with the respective poll.

4) Contract Address: The *Consortium App* also provides an auxiliary functionality in addition to the main voting operations. The form *HEI Contract Address* available on the interface, allows any consortium member to retrieve the contract address associated with a given HEI identifier. A member inserts the DID linked with a HEI and presses the *Submit* button, which will trigger a call from the application to the consortium contract. This request will call the *getHEI()* function from the contract, that receives a HEI identifier as an argument and returns the respective HEI contract address.

If the entered DID corresponds to a HEI that is part of the consortium and, therefore, registered in the consortium contract, the respective HEI contract address is displayed on the interface. Otherwise, it will present an error message indicating that the HEI is not a member of the consortium. This functionality is useful for members to check if a specific HEI is registered on the contract.

V. Evaluation

The solution is being evaluated in the context of our university (a HEI) and a public institute that often contracts our students. At this stage, we made performance measurements that we summarize below.

Ethereum is a permissioneless blockchain so anyone can join and use the infrastructure. To avoid abuse, Ethereum charges ether for the execution of smart contracts, which makes it inconvenient to use when the purpose is to evaluate applications. For that purpose there are a set of Ethereum test networks or *testnets* that emulate Ethereum's behavior and do not require payment (ether is obtained for free in a "faucet"). *Ropsten* was the testnet chosen since it is the one that most resembles the main network. Blocks are introduced at an average of 20 seconds on average, similarly to Ethereum.

TABLE I
Ecoystem evaluation results.

Operation	Throughput (cert/s)	Cost (ether)	Latency (s)
Register certificate	42.98	0.00023792	27
Revoke certificate	119.78	0.00013708	25
Verify certificate	74.27	0.00000000	1.285

We assess three performance metrics: *throughput*, operations executed per unit of time; *latency*, average time to complete operations; *cost* of executing the operation in Ethereum. The first two were measured using executions in Ropsten (repetitions of 100 operations, certificates of approximately 100 KB) and the third obtained in the Etherscan Ethereum Blockchain Explorer (that shows data about transactions and blocks in Ethereum and related systems). The first two involved running a modified version of the HEI client and the recruiter app. A more detailed explanation of these testing procedures is written below.

- *Register Certificate:* this operation was tested by using the *HEI Client* to register 100 education certificates to calculate the system's throughput. The measured latency expresses the interval of time between storing a certificate and the respective transaction being correctly sent to the blockchain network and added to a new block. After the transaction is confirmed, the transaction fee represents the amount paid to the miner for processing the transaction. The cost information can be obtained from a blockchain explorer such as *Etherscan*.
- *Revoke Certificate:* for this operation, 100 education certificates were generated to assess the system's throughput by utilizing the *HEI Client*. The latency metric indicates the interval of time between the request to revoke a certificate and the respective transaction being correctly sent to the blockchain network and added to a new block. The corresponding transaction fee represents the amount paid to the miner for processing the transaction. The cost information can be obtained from a blockchain explorer such as *Etherscan*.
- *Verify Certificate:* this operation was tested by generating and verifying 100 education certificates simultaneously through the *Recruiting App*. The latency presented represents the interval of time between the request for a certificate verification and the result of this verification (successful/failed).

The results are shown in Table I. Several interesting conclusions can be taken from the table. First, throughputs are reasonably low, but this was expectable as the experiments were done in a testnet and with a single client. Second, the latency for registration and revoking is apparently bad, 25s and more, but transactions are only executed after a block is inserted in the blockchain, something that happens on average every 20s. Third, the cost of verification is 0 and the latency is much lower than the others, because we are running a

local version of verification that runs in the local node (this is possible because this operation does not modify the state of the smart contract).

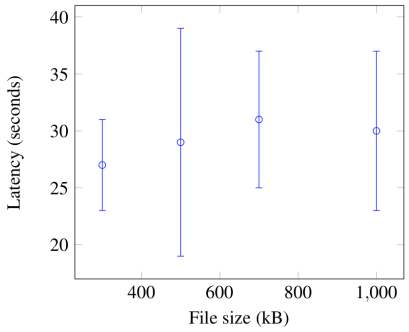

Fig. 3. Register Certificate latency for different file sizes.

In Figure 3 it is possible to observe that the certificate's size has little to no influence in the latency, since mining the respective transaction and adding it to a new block occupies the largest percentage of the processing time. The standard deviations correspond mostly to network delays, queue sizes and the Ether amount offered to the miners for processing a transaction.

From the obtained results, it is evident the benefits the adoption of this project can bring to any recruiting organization. Besides making recruitment processes more efficient, less time-consuming and cost free, it also gives employers the assurance of legitimacy and integrity in all education certificates received from their candidates.

VI. Software Engineering Challenges

As already mentioned, the major software engineering challenge we face is architectural: ensuring *decentralization*. A permissionless blockchain like Ethereum provides a high degree of decentralization by running in a large number of nodes without any central control (except for new software versions that, however, can be installed by the participants in their nodes). However our design shows that there are many traps – often observed in practice – that may impair that property:

- smart contracts may centralize *governance*, by being managed by a single organization (typically the entity that deployed it);
- web frontends may centralize *access*, by forcing requests to pass through a single web server (or a set of web servers under the same administration), in some cases even including a centralized backend database;
- file storage may be centralized in the cloud, network attacked storage, behind a web frontend, etc.

Solving these challenges involves keeping the need for decentralization in mind and scrutinizing the design looking for centralization, as this is the usual form of designing systems today. Decentralization is hard because it is not natural for today's system architects and programmers.

Our design suggests other challenges. A first example is *testing*, that has to be done locally before deployment, plus also later in runtime as the blockchain environment is complex. An intermediate scenario is running the system in a testnet, as we did in the evaluation.

A second example of challenge is *version management*. Smart contracts cannot be substituted by new versions, so version management functionality has to be implemented on the smart contract, e.g., by supporting an exception *newVersionAvailable* that returns the address of a new version of the contract. Again, governance decentralization is an issue: who decides about the new version?

VII. Conclusion

We presented the first ecosystem for certificate verification at a large, potentially world-wide scale. The ecosystem leverages blockchain for decentralization, as centralization is undesirable in such an heterogeneous context.

The ecosystem is based on a permissionless blockchain, currently Ethereum, that runs two types of smart contracts: *Consortium Smart Contract* and *HEI Smart Contract*. Moreover, three applications were developed, two for Higher Education Institutions and the other for recruiting organizations. The *HEI Client* allows an AMS to automatically register a freshly generated education certificate on the blockchain, while being relieved from the burden of such operations. The *Recruiting App* is able to verify a candidate's certificate authenticity by requesting registered information from the respective HEI's smart contract available on the blockchain. The *Consortium App* allows consortium members to interact with the voting system implemented on the consortium contract.

We concluded by discussing the software engineering challenges involved, mainly those related to decentralization.

The presented solution was developed and deployed in a real use case of Recruitment in the Public Sector Pilot, integrating a public school of engineering and technology, and also a public institute in the fields of administrative modernization, simplification and digital administration. This work is expected to be a baseline, from where other education institutions and employing organizations can adhere and build upon.

Acknowledgment

This work was supported by the European Commission program H2020 under the grant agreement 822404 (project QualiChain) and by national funds through Fundação para a Ciência e a Tecnologia (FCT) with reference UIDB/50021/2020 (INESC-ID).

References

[1] StatisticBrain, "Resume Falsification Statistics," https://www.statisticbrain.com/resume-falsification-statistics.
[2] R. L. Rivest, A. Shamir, and L. M. Adleman, "A method for obtaining digital signatures and public-key cryptosystems," *Communications of the ACM*, vol. 21, no. 2, pp. 120–126, 1978.
[3] ITU-T, "International Telecommunication Union. ITU-T Recommendation X.509: The Directory: Public-Key and Attribute Certificate Frameworks," 2000.
[4] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," 2008.
[5] S. Underwood, "Blockchain beyond Bitcoin," *Communications of the ACM*, vol. 59, no. 11, pp. 15–17, 2016.

[6] M. E. Peck, "Blockchains: How they work and why they'll change the world," *IEEE Spectrum*, vol. 54, no. 10, pp. 26–35, 2017.

[7] M. Correia, "From Byzantine consensus to blockchain consensus," in *Essentials of Blockchain Technology*. CRC Press, 2019, ch. 3.

[8] C. Kontzinos, O. Markaki, P. Kokkinakos, V. Karakolis, S. Skalidakis, and J. Psarras, "University process optimisation through smart curriculum design and blockchain-based student accreditation," in *Proceedings of 18th International Conference on WWW/Internet*, 2019.

[9] I. R. Keck, M.-E. Vidal, and L. Heller, "Digital transformation of education credential processes and life cycles: A structured overview on main challenges and research questions," in *12th International Conference on Mobile, Hybrid, and On-line Learning (eLmL 2020)*, 2020.

[10] A. E. Gencer, S. Basu, I. Eyal, R. Van Renesse, and E. G. Sirer, "Decentralization in Bitcoin and Ethereum networks," in *International Conference on Financial Cryptography and Data Security*. Springer, 2018, pp. 439–457.

[11] IMS Global, "OpenBadges v2.0," https://openbadgespec.org/, 2020.

[12] Blockcerts consortium, "Blockcerts," https://www.blockcerts.org/guide/, 2016-2019.

[13] E. Durant and A. Trachy, "Digital diploma debuts at MIT," http://news.mit.edu/2017/mit-debuts-secure-digital-diploma-using-bitcoin-blockchain-technology-1017, 2017.

[14] A. Mikroyannidis, A. Third, and J. Domingue, "Decentralising online education using blockchain technology," in *The Online, Open and Flexible Higher Education Conference: Blended and online education within European university networks*, Oct. 2019.

[15] CEF Digital, "European Blockchain Services Infrastructure (EBSI)," https://ec.europa.eu/cefdigital/wiki/display/CEFDIGITAL/EBSI, 2020.

[16] A. Norta, "Creation of smart-contracting collaborations for decentralized autonomous organizations," in *International Conference on Business Informatics Research*. Springer, 2015, pp. 3–17.

[17] C. Jentzsch, "Decentralized autonomous organization to automate governance," *White paper*, Nov. 2016.

[18] Ethereum team, "Ethereum: A next-generation smart contract and decentralized application platform," 2014, White Paper.

[19] N. Szabo, "The idea of smart contracts," http://szabo.best.vwh.net/smart_contracts_idea.html, 1997.

[20] J. Benet, "IPFS: Content addressed, versioned, P2P file system," *arXiv preprint arXiv:1407.3561*, 2014.

[21] Swarm, "SWARM: Storage and communication for a sovereign digital society," https://ethersphere.github.io/swarm-home/, 2019.

[22] S. Wilkinson, J. Lowry, and T. Boshevski, "Metadisk a blockchain-based decentralized file storage application," *Tech. Rep.*, 2014.

[23] Ethereum, "Solidity," https://solidity.readthedocs.io/en/latest/index.html, 2016-2019.

[24] W3C, "Decentralized Identifiers (DIDs) v1.0," https://www.w3.org/TR/did-core/.

[25] G. Tsudik, "Message authentication with one-way hash functions," *ACM Computer Communications Review*, vol. 22, no. 5, pp. 29–38, Oct. 1992.

[26] L. Luu, D.-H. Chu, H. Olickel, P. Saxena, and A. Hobor, "Making smart contracts smarter," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, 2016, pp. 254–269.

[27] A. Mavridou, A. Laszka, E. Stachtiari, and A. Dubey, "VeriSolid: Correct-by-design smart contracts for Ethereum," in *International Conference on Financial Cryptography and Data Security*. Springer, 2019, pp. 446–465.

[28] I. Medeiros, N. F. Neves, and M. Correia, "Automatic detection and correction of web application vulnerabilities using data mining to predict false positives," in *Proceedings of the International World Wide Web Conference*, Apr. 2014, pp. 63–74.

[29] T. Llanso and M. McNeil, "Estimating software vulnerability counts in the context of cyber risk assessments," in *Proceedings of the 51st Hawaii International Conference on System Sciences*, 2018.

[30] T. Durieux, J. F. Ferreira, R. Abreu, and A. P. C. Monteiro, "Empirical review of automated analysis tools on 47,587 Ethereum smart contracts," in *42nd International Conference on Software Engineering (ICSE 2020)*, Dec. 2019.