ShareCert: Sharing and Authenticating Certificates and Credentials on Blockchain

Vasista Sai Venkata Durga Prasad Lodagala
Department of Mathematics and Computer Science
Sri Sathya Sai Institute of Higher Learning
Anantapur, Andhra Pradesh, India
vasista96@gmail.com

Pallav Kumar Baruah
Department of Mathematics and Computer Science
Sri Sathya Sai Institute of Higher Learning
Anantapur, Andhra Pradesh, India
pkbaruah@sssihl.edu.in

***Abstract*—**The process of certificate and credential verification of every candidate is an essential step in the recruitment phase of every organization. The employer needs to verify if the certificates provided by the candidate are authentic. Today, most of this work is generally outsourced by the organization to external agencies who perform the verification at the behest of the organization. Owing to the presence of third parties and lack of transparency in the verification process, there is a possibility of fraud or undetected error creeping in at every stage. The organization is at a risk of hiring a fraudulent candidate and the candidate at the risk of getting rejected due to inefficient verification. Moreover, the existing procedure for credential verification takes around 30 days for a candidate.

Blockchain technology is one of the few innovations in computer science that has surpassed the fame of its initial application which is the bitcoin cryptocurrency. Though Bitcoin cryptocurrency was the first to introduce blockchain technology as its underlying strength, the varied applications of blockchains is eventually surpassing the buzz around cryptocurrencies. Smart Contracts is one of the key contributions of the technology. Today, applications developed using Blockchain technology ranges from Supply Chain Management to Digital identity, Internet of Things, Decentralization over control of data available on the internet, Banking, Insurance and many(any) other scenarios where multiple entities form a network to trade or query any kind of assets.

Using smart contracts, a decentralized data store, content-hash based verification and blockchain, we've developed decentralized applications on blockchain platforms such as Ethereum and Hyperledger Fabric to address the existing issue of certificate and credential verification. These decentralized applications bring about transparency, privacy, provenance tracking and trust to the procedure of sharing and authenticating credentials and certificates. In terms of efficiency, our framework supports real-time verification of certificates and credentials, thereby providing a faster and transparent verification mechanism.

Keywords—Certificate Verification, Blockchain, Smart contracts, Ethereum, Hyperledger Fabric

I. MOTIVATION

Civilization today is dependent on efficient operation of various Private and Public organizations. These organizations perform based on the individuals working for them to run the affairs in an efficient manner. A key phase of running an organization efficiently is the recruitment process. An important aspect of this process is to verify if the certificates produced by the candidate are authentic. This is because there are cases of individuals submitting fake certifications. The existing process of educational certificate verification involves the following steps:

Figure 1

This entire procedure involves centralized organizations and the participation of trusting third party organizations. The existing procedure is slow and involves limited participation from the candidate in the validation process.

A. Limitations

We discuss the limitations of the existing methodology from the perspective of each of the three entities.

1. Candidate:
    a. After the submission of certificates to the employer, there is no participation of the candidate in the verification procedure.
    b. It is not transparent. The entire verification procedure is carried out by the employer and the external agencies that work for the employer. In general, the details of the verification are kept confidential by the employer.
    c. Because the candidate has no say in the verification procedure, there are no means by which he/she can ascertain the legitimacy of the verification process.
    d. The current methodology doesn't consider the privacy concerns of the candidate. Since candidate's personal data such as certificates and transcripts are handled by external agencies, it could lead to data breaches. Again, the candidate would have no idea who has initiated the privacy breach.

2. University:
    a. With the total number of candidates who have graduated from the university increasing every year, the number of potential verifications that needs to be done increase with every passing year.
    b. As the current procedure isn't completely digital, the university would need to employ additional manpower to handle the verification.
    c. There could be internal fraud happening within the university that could facilitate validation of false records and certificates.
    d. The process involves a lot of human handlers which renders it error prone.
3. Employer:
    a. The existing validation methodology is an overhead for the employer at the time of recruitment of candidates into the organization.
    b. The existing verification methodology isn't cost effective as it involves costs of transportation, verification fee, etc. With a large number of candidates to be recruited, a company would need to invest a considerable amount to finance the verification process.
    c. The current procedure for verification takes around 30 days per candidate, which is a considerable amount of time for which the employer needs to put the recruitment on a hold.
    d. The employer needs to place trust on middlemen which are agencies that perform the actual verification procedure.

As we can notice, the entire validation procedure contains risks and limitations for all the three entities. The core threats in the current system are lack of privacy, lack of transparency, trusting middlemen and centralization of verification procedure (only the university is involved in the final verification process). Also, on an average the existing procedure takes up to 30 days for credential verification per candidate.

B. Possible Solution

After knowing that the existing methodology is time-consuming, it is a no-brainer that making the entire process digitally driven would drastically bring down the verification time. A possible solution in this direction would be to use a centralized database to Create, Read, Update and Delete records (CRUD operations).

Though the solution of using a centralized database helps in improving the speed of verification, it fails to address most of the limitations of the existing methodology. This is because, the solution using a centralized database doesn't guarantee content privacy and transparency of the verification process as the verification process can still be outsourced to external agencies.

For instance, the February of 2018 witnessed one of the biggest Distributed Denial of Service (DDoS) attacks on GitHub, a code management platform popular among developers. It was a Memcached DDoS attack, so no botnets were involved. the attackers leveraged the amplification effect of a popular database caching system known as Memcached. By flooding Memcached servers with spoofed requests, the attackers were able to amplify their attack by a magnitude by about 50,000 times. At its peak, this attack saw incoming traffic at a rate of 1.3 terabytes per second (Tbps), sending packets at a rate of 126.9 million per second. [1]

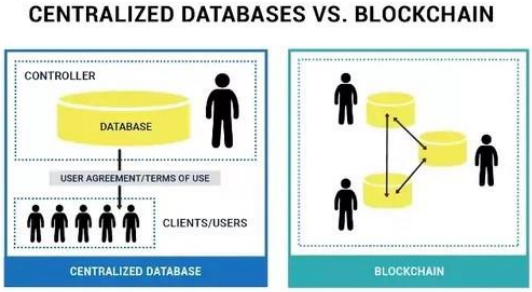

Figure 2 *[2]*

At this juncture, we propose to resolve this verification issue in the most efficient manner using a blockchain, smart contracts and a decentralized data store, all of which are detailed about in the following section.

II. INTRODUCTION

In this section we introduce blockchain, smart contracts and decentralized data stores which form the fundamental structure for developing our solution to share and authenticate certificates and credentials in a decentralized, transparent and privacy preserving manner.

Decentralization of power and transparency in governance are two of the most sought after organizational and societal principles. These are the fundamental principles that win the trust and confidence of the participants of any organization or society. Blockchain technology by design, weaves both these principles together with the additional features such as preserving user privacy and immutable record keeping.

Blockchain was first popularized by Satoshi Nakamoto through the introduction of Bitcoin [3]. The strength of blockchain lies in the fact that it is immutable and provides privacy and transparency. Now, the idea of both privacy and transparency being provided at the same time might seem contradictory. But here, the privacy preservation refers to anonymizing the users(entities) that are a part of the network. On the other hand, transparency is with respect to transactions. All of the transactions made on blockchain are generally made public. So blockchain ensures that transaction data is transparent to all but at the same time doesn't reveal the sender and receiver explicitly.

In layman terms, blockchain can be thought of as a secure immutable way of storing any type of data ensuring privacy and data integrity. A more technical definition would be: Blockchain is a distributed ledger technology with some or all entities in the network having the permission to write to the ledger, some or all entities in the network requiring to agree upon the distributed ledger and all the entities in the network having the permission to read or query the ledger. Simply put, blockchain is a decentralized, distributed and immutable database preserving user privacy.

A blockchain can also be thought of as a linked list of hash pointers [4]. A hash pointer points to the location where the data is stored and also contains the hash to the value of the data. A block in a blockchain is simply a bundle of transactions that are validated by the peers in the decentralized network. Blockchain is also referred to as ledger. This ledger is updated and kept consistent among the peers through a consensus mechanism on the decentralized network. Each block in the blockchain contains the cryptographic hash of its previous block. This method of chaining blocks using cryptographic hashing is what provides blockchain its immutability. This is because, if an adversary has to alter the contents of a transaction in a block that already exists on several peers, it would also lead to the modification of the associated block because, it is the transactions that make up blocks in a blockchain. Once a block is modified, the hash of that block also changes, and would not match with the hash stored in the subsequent block in the ledger. In order to succeed, the attacker needs to modify all subsequent blocks that are added to the blockchain. This is computationally infeasible, especially in a blockchain network that implements Proof-of-Work (PoW) as the underlying consensus mechanism.

A. Computation - Record Keeping - Storage

In today's world where data is becoming increasingly valuable, privacy breaches are not quite uncommon. Assisted with efficient record keeping mechanism that is immutable and preserves user privacy, blockchain technology offers a trustable way to handle sensitive user data. These properties bring value to user data. So, we could refer to user data as assets that are a part of the blockchain. Essentially it is the trade of these assets that brings about blockchain applications in varied fields such as banking, insurance, health care, social networking, credential authentication, voting, land ownership records, file sharing, etc.

Though blockchain can host the data (hash of the data) necessary for the functioning of the mentioned applications, we need efficient programs to perform computations on this data. But if these programs are not a part of the blockchain, then the entire exercise of keeping data on blockchain becomes pointless. So, these programs are stored on the ledger and are referred to as "smart contracts". Smart contracts contain the business logic, that is agreed upon by all the parties involved in the business network. This business logic is executed once certain transactions are triggered by certain entities in the network. The code embedded in a smart contract is executed by every node in the blockchain providing redundancy in computation leading to very high reliability and almost zero downtime. Also, the execution of these smart contracts happens only when certain conditions are met.

As discussed earlier, blockchain is an effective tool for record keeping. Record keeping as known by all just involves storing the details of an asset rather than the asset itself. Similarly, when blockchain is used as a record keeper of user data, decentralized data stores are used to store the data itself. We emphasize on decentralized data stores because, centralization of data could lead to multiple issues such as: single point of failure, Denial of Service (DoS) attacks and data tampering [2].

Essentially, while smart contracts execute transactions (computation), blockchain logs these transactions (record keeping) and decentralized data stores such as the Interplanetary Filesystem (IPFS) [5] provide the necessary space(storage).

We'll now delve into certain details of the blockchain frameworks which we worked on.

B. Frameworks

With the help of blockchain, smart contracts and decentralized data stores, we've developed two designs to solve the certificate verification problem which we've implemented across two popular blockchain platforms: Ethereum and Hyperledger Fabric.

Before we actually discuss our designs and implementations, we introduce the purpose and functionality of Ethereum and Hyperledger Fabric.

1) Ethereum

While the application of blockchain to bitcoin and other cryptocurrencies is treated as Blockchain1.0 [6], the Ethereum platform co-founded by Vitalik Buterin brought about a revolution in the way blockchain is applied for computation [7]. This revolution brought about by Ethereum is referred to as Blockchain 2.0 [6]. This is because Ethereum supports distributed data storage and computation, along with providing a native cryptocurrency Ether, as a part of its implementation. The greatest contribution by the Ethereum platform is that it utilizes blockchain technology to develop decentralized applications. Such decentralized applications, which have smart contracts as the basis, are executed on every node in the Ethereum network without any possibility of downtime, censorship or third-party interference. Consensus is reached among nodes using the Proof-of-work algorithm to update the global state upon execution of smart contracts. Ethereum embodies the idea of single blockchain running any arbitrary computation. Embedding business or transaction logic as a part of smart contracts ensures its correct execution, thereby preluding any possibility of fraud. Smart contracts help in significantly reducing the business costs involved and helps run de-risked businesses. Using the Ethereum platform, a developer can create numerous decentralized applications which could include other cryptocurrencies as well.

Contracts are located at a specific address in the chain and transactions can be sent to this address to execute the contract. A contract in Ethereum includes code as well as data. In addition, contract accounts can also pass messages to each other to execute necessary contract functions or access data. These contracts are executed by the Ethereum Virtual Machine (EVM) which is run by every node that is a part of the Ethereum network. Contract code is stored in the Ethereum blockchain in a binary format called the EVM bytecode. To facilitate contract writing, many high-level languages are provided. Contracts written in such languages are then complied into EVM bytecode which can be stored on the blockchain. Also, an intermediary JavaScript Object Notation (JSON) file can be created from the contract bytecode to be deployed locally. JSON is chosen because it can describe the functionality of the contract and still remain human-readable. The most common language for writing smart contracts is Solidity which is similar to JavaScript.

It is the EVM that executes the smart contracts. **Gas** is the unit which is used to measure the EVM resource usage. Gas usage in a transaction depends on the number of instructions required by the EVM to execute the transaction and the storage space used by the transaction. Each operation (OP) code in the EVM has fixed units of gas associated with it. The miners who are executing the transaction, are spending some hardware resource for computation. To provide incentive for the miners, a transaction fee is paid as in the case of bitcoin. Unlike bitcoin, transaction fee is computed based on a gas price fixed by the user before submitting the transaction.
User defines two parameters before submitting a transaction. They are:

- Start Gas (units): Maximum units of gas, the originator of the transaction is willing to spend.
- Gas Price (ethers): per unit gas price the originator is willing to pay.

Fee processing: Before a transaction gets validated, the maximum amount of ether the user is willing to spend is kept in an escrow. So, the escrow contains the amount of ether which equals the product of Start Gas and Gas Price. Fee that is paid to the successful miner is determined as the product of Gas Used and Gas Price. Any additional ether is refunded to the user.
Successful transaction executions update the Ethereum state, which includes the balances of the Externally Owned Account (EOA), and the balances, code and data of the contract accounts. In case the initially specified amount of ether isn't sufficient for the computation, an *Out of gas exception* is specified to the originator and the transaction is rolled back. However, the ether in the escrow is given to the miner as a reward for the computation performed.

2) Hyperledger Fabric

Hyperledger Fabric is one among the many open source blockchain platforms being developed by The Linux Foundation under the umbrella of Hyperledger projects. As stated in the Hyperledger project, only an open source collaboration can ensure the transparency, longevity, interoperability and support required to bring blockchain technologies forward to mainstream commercial adoption, and it is this collaboration that is the goal of the Hyperledger project [8].
Fabric aims to provide the base architecture for custom enterprise networks. It is modular, allowing for pluggable implementation of blockchain components, such as smart contracts (referred to as chain code), cryptographic algorithms, consensus protocols and data store solutions. It can be deployed using container technologies and provides a secure chain code execution environment that protects the network against malicious code. It is a robust and flexible blockchain network architecture with enterprise-ready security, scalability, confidentiality and performance.
Key design features of Fabric are:

- ✓ It provides the necessary support to develop private permissioned blockchains, as required by most of the business applications.
- ✓ In order to improve the privacy and confidentiality of assets and transactions, Fabric supports the notion of channels. Only those entities that are a part of the channel can view the transactions recorded on the ledger.
- ✓ It provides identity services in the form of Certificate Authorities and Membership Service Providers (MSPs).
- ✓ The immutable, shared ledger records the entire transaction history for each channel and supports SQL-like queries for auditing and resolution of disputes.
- ✓ Modularity in Fabric enables customization of the Fabric for the specific use-case thereby improving efficiency of the blockchain network against general, fit-all but specialized-for-none networks. The network designers can choose the specific consensus, identity management and encryption algorithms to plug-in to suit their needs.
- ✓ Transaction processing in most of the blockchain platforms is order-execute. That is transactions are ordered and blocks are formed first, following which the transactions are executed and the validation of transactions are performed by rest of the peers. This leads to significant downtime because, forming the block itself takes time in case of consensus mechanisms such as PoW. Fabric on the other hand adopts a execute-order-validate strategy, which aids in improving the transaction processing speed.

The Fabric architecture comprises of peer nodes, ordering nodes, client applications and chain codes that can be written in any of the supported Fabric Software Development Kits (SDKs). Currently, there are SDKs for Go, Node.js, Java and Python. Fabric also offers a certificate authority service which, following its modularity principle, can again be replaced by custom implementations.
There are two places where data is stored in a Fabric network. While one is the immutable ledger (blockchain), the other more important place is the world state. The world state holds the current value of the attributes of a business object as a unique ledger state. That's useful because programs usually require the current value of an object; it would be cumbersome to traverse the entire blockchain to calculate an object's current value – you just get it directly from the world state. This state is stored in the form of key-value pairs. Level DB is the default database used for storing the world state. CouchDB can also be used in its place to support complex and rich JSON queries.
Nodes in a fabric network have distinctly assigned roles. There are 3 types of nodes in a fabric network. They are peers, endorsers and orderers. All the peer nodes are committers and are responsible for maintaining the ledger by committing transactions. Some nodes are also responsible for simulating transactions. They execute the chaincodes and endorse the result, and are called endorsers. A peer can be an endorser for certain transactions and just a committer for others. The responsibility for ordering the transactions in a block to be committed is on the orderers. The roles of committers and orderers are generally unified in common blockchain architectures.

III. Literature Survey

In the literature we find two existing works similar to ours. While one of the designs is by a set of two organizations

named Serokell and Disciplina, the other implementation has been developed at the Massachusetts Institute of Technology (MIT)'s Media Lab and bears the name Blockcerts.

A. Disciplina

Disciplina is a platform that is designed to act as a decentralized ledger and has a special regard to privacy mechanisms and data disclosure [9]. As a platform, Disciplina aims to transform the way educational records are generated, stored and accessed.

The major requirements for a platform like Disciplina are:

i. The platform should support storage of large quantities of private data such as grades, assignments, solutions, etc.
ii. Only buyers/employers must be the entities to whom data can be disclosed to.
iii. There should be no third-party interference in the verification process and the platform must support fairness of data.

The design choice for using a hybrid blockchain has been made based on the following rationale. Though we can use a public ledger to store data after encryption, it suffers from incentive and scalability issues because nodes in the network must be able to store the records provided by educational institutions from all over the world. Therefore, the use of public blockchains for the verification procedure is economically unjustified.

1) Architecture

There are four entities involved in the entire validation process. They are namely, *candidates, educators, recruiters and witnesses*. A private chain is maintained between the candidates and educators. A public chain is maintained between the recruiters and witnesses.

We must note that the private layer is maintained by each educator independently. All the interactions between the educator and candidates such as receiving assignments, submitting solutions or being graded are treated as transactions on the private chain. This private chain can be accessed by the candidate through a web or mobile application.

The verification process can be captured in the following picture:

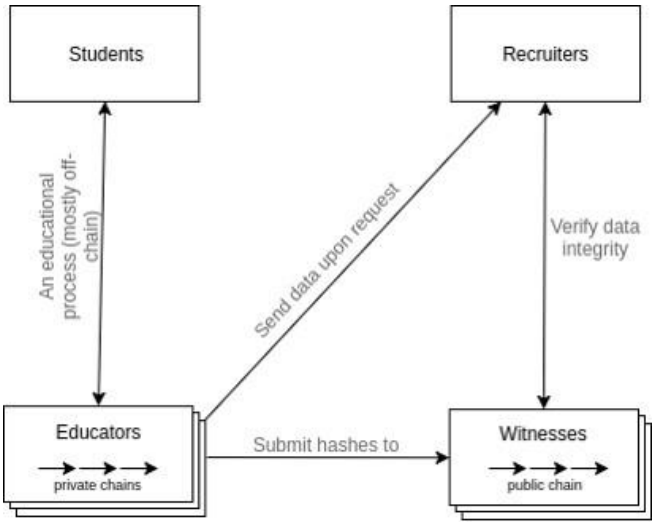

Figure 3 [9]

An elaborate discussion of the verification process from the platform's perspective is as follows:

i. Candidate would enrol in a course of his or her choice after choosing an educator.
ii. Candidate must have completed the fee payment formalities if the course is a pre-paid one.
iii. As the course progresses, the educator gives assignments to the candidate, which are to be completed by the candidate, in order to receive a valid score.
iv. Upon acquiring the assignment from the educator, the candidate submits a signed solution of the assignment to the educator. This candidate-educator communication happens completely off chain.
v. After grading the solution locally, the educator transfers the score along with its hash onto the private blockchain.
vi. After the course completion, the candidate acquires a final score, which is also added onto the educator's chain.

Though there exists a private chain maintained by the educators, the is no possibility of tampering with the transactions as all the private transactions are made publicly verifiable. In this way a second public layer of blockchain is introduced.

This public part consists of *witnesses*. These are the nodes that witness a private block produced by the educator. This verification of private blocks is done by the witnesses by writing the authentication information of every private block onto the public chain. This data on the public chain can be used in the future by any arbitrary verifier to substantiate the proof of transaction inclusion given to it by the candidate or the educator.

Also, witnesses process public information that is issued by the educators. This information for example could be that, an educator has stopped offering a particular course. Recruiters can buy candidate data from educators.

B. Blockcerts

It is an open infrastructure for sharing academic credentials on the Bitcoin blockchain. Blockcerts provides a decentralized credentialing system. The bitcoin blockchain acts as a provider of trust, and credentials are tamper-resistant and verifiable. Blockcerts can be used in the context of academic, professional and workforce credentialing [10].

There are four components involved in the Blockcerts implementation:

i. Issuer: Digitalized academic certificates are created by the universities and can contain an array of attributes describing the individual's skill sets and achievements. These certificates are registered on the bitcoin blockchain.
ii. Certificate: open badges is the standard followed to create certificates because that is becoming the globally accepted standard for digital certificates.
iii. Verifier: Any node can verify if a certificate is tamper-proof and whether it has been issued by a particular institution and to the specified user.

iv. Wallet: is used by individual entities to safely store their certificates and share with other entities such as an employer.

1) Design

The following three repositories form the architecture:

1. Cert-schema: This describes the data standard to create and manage digital certificates. The digital certificate is a JSON document having the necessary fields required for cert-issuer code to be stored on blockchain. The schema is similar to that of open badges specification.
2. Cert-issuer: Here, the hash of the digital certificate is generated and broadcasted as a bitcoin transaction, from the issuer's address to the recipient's address with the hash being a part of the OP_RETURN field.
3. Cert-viewer: used to display and verify the digital certificates that have been issued. It also provides users the permission to request certificates and generate a new ID on bitcoin blockchain.

This design uses asymmetric key cryptography to authenticate the issuer and the recipient. Revocation of certificates is done through a transaction on blockchain by setting a flag that says that a particular certificate is invalid.

2) Advantages

- Candidates own their records.
- Vendor-independent verification of blockcerts.
- The advanced version of blockcerts can issue certificates to any blockchain platform. However, this platform is still under development.
- Portability of certificates: because open badges standard has been used.
- Private: transactions are private and only a digital fingerprint is stored on the blockchain.
- Interoperable: due to the open-badges standard adopted.
- Open source.

Having explained the existing solutions now we will describe our solution for the certificate and credential verification in the next section.

IV. Proposed Designs

After reading about the functionality and features that blockchains provide, we can conclude that sharing and authenticating certificates and credentials on blockchain is a perfect application of smart contracts. With the introduction of blockchains, user privacy is maintained as the identity of the user is masked with the help of Public Key Identities (PKI). As any node can view the transaction history, transparency is brought into the verification procedure. Dependency on the services of middlemen is eliminated, because the entire process is performed with the help of smart contracts.

With the adoption of blockchain technology to design an efficient solution, we make the entire verification process transparent. Our implementation focuses on bringing the responsibility onto the candidate to initiate the verification procedure. This reduces the overhead on the employer to take up the verification procedure. Also, no middlemen need to be entrusted with the responsibility of taking up the verification process.

An overview of our solution using blockchain is given in the bulleted points:

- Certificates are cryptographically secured and shared between the three entities through a decentralized data store. Each data exchange is recorded as a transaction on the blockchain for verification.
- Once recorded on the blockchain, transactions cannot be tampered with and are distributed across the nodes involved in the validation process.
- Verification of certificates and credentials doesn't involve any central authority or middlemen. Verification is done in real-time through smart contracts, used to build the decentralized application.
- A salient feature of our proposed model: The interaction between the university and the employer is now made optional. The complete verification is done through a transaction triggered by the candidate.

The smart contract which forms the backbone of our decentralized application is what interacts with the underlying blockchain. Though the decentralized data store contains the certificate, the certificate is encrypted and then uploaded to the decentralized data store. This ensures that privacy of the candidate is taken into consideration. Even though the decentralized data store is public, with anybody having access to the files, this kind of encryption mechanism, prevents access to the contents of the file.

We propose two designs as a part of this work, both of which are discussed in detail in the following subsections.

A. Design - 1

Each important phase in the complete procedure of certificate verification is present as a transaction, which the smart contract supports. The smart contract is a collection of such transaction processing functions. The **key phases** of the entire process can be listed as follows:

1. (Transaction 1 / Request): Candidate requests the certificate from the university by providing the necessary inputs such as the year of graduation, registration number and other relevant metadata.
2. (Transaction 2 / Response): University reviews the candidate's request and publishes the encrypted digital certificate on a decentralized data store. This encryption is done using the public key of the candidate, which is available to the university either at the time of candidate enrolment or through interaction on the smart contract. The decentralized data store returns the hash of the encrypted certificate. The address(hash) of the certificate is shared with the candidate as a part of the response.
3. Candidate decrypts the certificate using his private key, after downloading a copy of the encrypted certificate from the decentralized data store.
4. (Transaction 3 / Validation): Candidate shares the certificate with the employer after encrypting it using the employer's public key and publishing it on

the decentralized data store. Along with this, the candidate sends the transaction ID of the response he/she received from the university. The employer encrypts the certificate using the candidate's public key and obtains the hash for the file from the Application Programming Interface (API) offered by the decentralized data store. The employer then checks this hash against the hash sent by the university to the candidate. Employer has access to the later hash from the transaction ID of transaction 2 shared by the candidate with the employer. Post verification, the employer then responds to the candidate appropriately about the state of validation. This interaction from the employer to the candidate forms (Transaction 4).

B. Design - 2

This approach is completely permissioned. It defines the permissions for each entity specifically. For example, a candidate can only view those requests and responses that he's sent or received to/from the university. An employer can only see the requests and responses of a candidate after the candidate has initiated the verification process that also involves the employer.

1. (Transaction 1 / Request): Candidate requests the certificate from the university by providing the necessary inputs such as the year of graduation, registration number and other relevant metadata. He does this by creating an asset called *request* which would contain the above-mentioned inputs along with the requestID. This asset is sent to the university through a send request transaction.
2. (Transaction 2 / Response): University reviews the candidate's request and publishes the encrypted digital certificate on a decentralized data store. This encryption is done using the public key of the candidate, which is available to the university either at the time of candidate enrolment or through interaction on the smart contract. The decentralized data store returns the hash of the encrypted certificate. The university creates a *response* asset that contains the hash of the published certificate as returned by the decentralized data store. This asset contains a field called responseID with which this response can be identified when queried. Also, as a part of this asset, the university would specify the name of the degree or diploma awarded and the class obtained. This asset is then sent by the university to the candidate in a send response transaction. This send response transaction contains a reference to the request to which the university is responding and another reference to the response generated.
3. Candidate decrypts the certificate using his private key, after downloading a copy of the encrypted certificate from the decentralized data store.
4. (Transaction 3 / Validation): Candidate creates an asset called *proof* that would contain the requestID of the request sent by the candidate to the university and the responseID of the response sent from the university to the candidate. Also, the proof asset would contain the hash of the encrypted certificate which the candidate mentions. This proof contains a field called check hash status which cannot be set by the candidate. Once the candidate submits this asset called proof as a part of the send proof transaction, the underlying smart contract checks the hash of the certificate as mentioned by the candidate against the hash of the certificate as mentioned in the response from the university. Check hash status is a Boolean field which would get set based on the result of this verification. Since this transaction is addressed to the employer, the employer can view the result of the verification. Also, after this transaction, the employer would have the permissions to view the request, response and the proof. This way, the employer can be convinced of the validation of certificates, without even viewing them.
5. There is no need for another transaction between the employer and the candidate, as the result of the verification process is transparent to both these entities. The advantage of this approach is that, there is no need for the university to explicitly be a part of the verification process. The credentials of the candidate can be viewed by the employer, who gains read access to the response asset after the send proof transaction. There is no need for the employer to see the contents of the certificate because the verification is done through a smart contract which all the parties trust. Also, the employer has access to the response asset which contains the credentials and the degree or diploma awarded to the candidate. Therefore, as a part of this second design, sharing the contents of the certificate between the candidate and the employer is made optional. We say, the contents of the certificate because, though the employer has access to the hash of the certificate on the decentralized data store, he or she doesn't have access to the contents because the certificate is encrypted using the public key of the candidate. Another salient feature of this verification procedure is that, the entire process of certificate and credential verification is initiated and completed by the candidate.

Our work essentially offers two diverse designs that could complete the verification process using smart contracts, blockchain and decentralized data stores. We emphasize that, sharing of the certificate between candidate and employer need not as a part of the implementation of the second design because, this approach uses a permissioned blockchain that would limit the access to the assets. We'll discuss this part in further detail in a later section that talks about the implementation on Hyperledger Composer.

All the transactions are logged on the blockchain which functions as an immutable record and can be used for audit purposes anytime using simple queries. Also, the transactions are visible to all the entities in either of the designs. The three entities would need to agree upon the smart contract that would be used as a part of the verification procedure.

The following picture provides a non-technical view of the entire procedure:

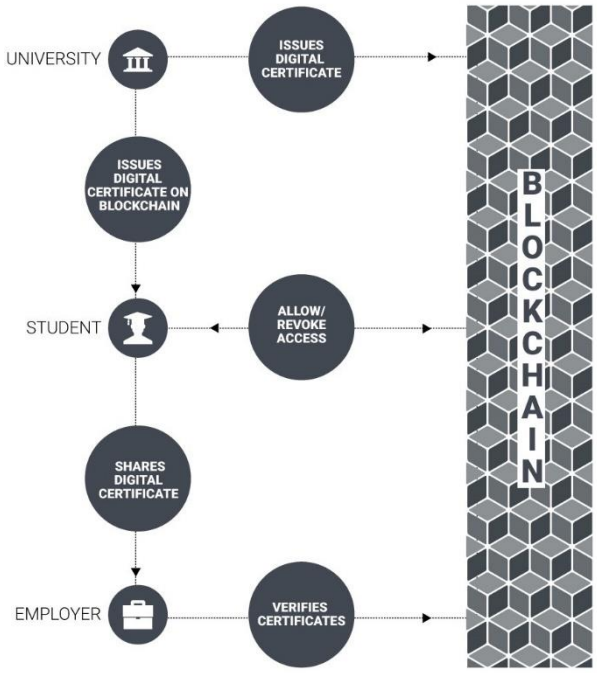

Figure 4 [11]

V. Implementations

As mentioned earlier, we've proposed two designs based on the method opted to perform the certificate verification as a part of the third transaction. Both these designs have been implemented as a part of this work. While we've implemented the design 1 on Ethereum, we've implemented the second design on Hyperledger Composer.

The choice for these two platforms has been made after taking into account the features offered by these two platforms and the architecture of the two platforms. While Ethereum is a public permission less blockchain, with any node having the permission to be a part of the blockchain and participate in the consensus procedure. Hyperledger Composer essentially is a Software Development Kit (SDK) used to interact with Hyperledger Fabric. Hyperledger Fabric is a platform that supports to maintain any kind of ledgers, public-permissioned, public-permission less, private-permissioned and private-permission less blockchains. But among developers, Fabric is popular in maintaining private-permissioned and private-permission less blockchains. Further justification of the choice of blockchain platforms for each design implementation is provided in the Results section.

We'll now explore the two implementations in greater detail by describing how the design has been implemented on each of these platforms.

A. Ethereum

Being the most popular smart contract platform, Ethereum popularized the use of smart contracts for business application purposes. Many decentralized applications were developed on top of Ethereum which is public-permission less in its design. Our implementation on Ethereum follows the point 4 which was mentioned in the key phases of the transaction process.

The smart contract used for the underlying decentralized application was first written and tested on Remix IDE which is available online. The smart contract was published through a transaction on the Ethereum network. As the Ethereum main net is not used for development purposes, we've deployed our smart contract onto the Ropsten test net.

The three entities interact with each other using a browser-based application developed using ReactJS, which in turn provides the interface for interaction with the smart contract. The actual interaction of the decentralized application with the underlying smart contract is facilitated with the help of web3.js, which is a collection of libraries that supports interaction with local or remote Ethereum node, using a HTTP or IPC connection.

Openpgp.js which is a JavaScript library supporting asymmetric key cryptography is used for the encryption of the certificates before uploading to IPFS. This ensures that no participant other than the designated candidate is given the permissions to view the contents of the certificate without access to the private key.

Storage on Ethereum is expensive as the EVM resource usage needs to be paid for. So, we've chosen IPFS as the decentralized data store for uploading our encrypted certificates. Ipfs.js is the JavaScript library used by the decentralized application to upload the encrypted content directly onto IPFS.

A simple JavaScript library named download.js has been used to facilitate the download of the file uploaded onto IPFS. This reduces the time taken to download the file from the daemon or the browser interface provided by IPFS.

The entire development of the decentralized application can be summarized using the following diagram:

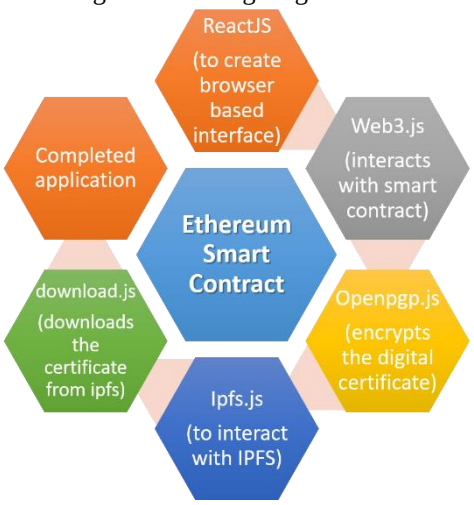

Figure 5

The novelty provided by our implementation is that, the employer can verify the authenticity of the certificate without any need for interaction with the university. This saves around 30 days of time spent for verification, as the verification supported by application is done on a near real-time basis.

Now, we describe the entire procedure to see, how our Ethereum implementation caters to the design proposed.

i. Candidate requests the certificate from the university using a transaction, by calling a function named *sendRequest* as defined in the smart contract. This function takes the candidate's name,

registration number, batch number, reason-for-request and a field called metadata as input. The field meta data is used to specify any other relevant data that the candidate wishes to provide. All the fields are of type string. The request is sent to the Ethereum address of the university.

ii. The university reviews the request, and generates a response after encrypting the certificate using the candidate's public key and uploading the file onto IPFS. It then sends the response through a transaction, by calling the *sendResponse* function as defined in the smart contract. The function takes, candidate's name, registration number, batch number, transaction ID of the request transaction, response-comments, and the IPFS hash fields as the inputs. All the fields are of type string. The response is sent to the Ethereum address of the candidate.

iii. Candidate downloads the file and decrypts it using his or her private key.

iv. Candidate encrypts the certificate using the public key of the employer, following which he or she uploads the file onto IPFS. Candidate then generates a transaction called *sendProof* which takes the candidate's name, registration number, IPFS hash of the uploaded file and the transaction ID of the response received from the university as the inputs. All the inputs are of the type string. The proof is sent to the Ethereum address of the employer.

v. Upon decrypting the certificate received from the candidate using his private key, the employer encrypts the certificate using the public key of the candidate. Using the API provided by the ipfs.js library, the employer computes the hash of this encrypted file. The employer then compares this hash with the hash present in the transaction mentioned in the sendProof transaction. This way the employer is verifying the certificate received from the candidate with the certificate sent to the candidate from the university to the student.

vi. Based on output of the hash verification, the employer notifies the candidate using a transaction that calls the *notifyCandidate* function.

This six-step verification procedure is what is used in the implementation on Ethereum. As we can see from the steps mentioned above, this way of verification completely eliminates the direct interaction between employer and university. This is a great benefit because, it makes the entire verification process driven by the candidate directly and is a completely transparent process.

B. Hyperledger Composer

Hyperledger Composer is a framework developed under the umbrella of open source Hyperledger projects by The Linux Foundation [12]. This framework is helpful in accelerating the development of blockchain based business applications on Hyperledger Fabric. Like fabric, composer supports modular approach in building decentralized applications. Though Composer is a separate project being developed by The Linux Foundation, it can be thought of as a Software Development Kit (SDK), which has the potential to build decentralized applications in a faster and more efficient manner.

One must realize the difference between the other SDKs such as node SDK, python SDK etc., that are supported by Fabric and Composer. Compared to those SDKs, Composer brings ease to the process of application development by auto-generating the chain code based on the configuration and business logic specified by the developer. Also, it brings down the direct interaction with the underlying Fabric network, by providing necessary abstractions where the developer can define the different sets of participants, assets, transactions, permissions, business logic and queries.

Apart from local development, Composer also provides an online playground making it even faster for developers to test code and build queries. Hyperledger Composer serves requests using a Representational State Transfer (REST) server with which the decentralized application interacts.

In contrast to development on Ethereum, development of smart contracts on Composer is completely modular. Though Ethereum offers modularity in usage of front-end libraries to build the complete decentralized application, Composer helps in modularizing the smart contract itself. Development of applications on Composer involves writing separate files which together form a business network archive which in turn interacts with the underlying fabric network.

A business network archive on Composer contains:

- Model file (.cto): This file is used to define all the assets, participants and transactions that would be a part of the application. Assets carry value and are traded on the network between participants using transactions.
- Script file (.js): This file contains the transaction functions that would be invoked upon submission of transactions by participants in the network. Updating assets is done in this phase, using transaction functions. Also, transfer of ownership happens with the help of these functions.
- Access Control (.acl): As we've mentioned earlier, Composer and Fabric are helpful in developing permissioned blockchains. This file contains the access control rules which are used to assign permissions to different entities involved in the application. A typical rule in this file has fields such as description of the rule, participant to whom the rule applies to, operation allowed, resource to which the rule applies and the actions permitted by the rule.
- Query file (.qry): This query file defines all the possible queries that a user is permitted to execute. These queries by design obey the access control rules, that is, a query cannot be run on a resource to which the user doesn't have access to.

All the above-mentioned files are bundled into a business network archive that can be deployed either on local Hyperledger Fabric environment or the online playground. The front-end interface for this implementation was developed using AngularJS whose code is auto-generated using an open-source client-side scaffolding tool for web applications named Yeoman.

In order to issue and store certificates, we use IPFS as the decentralized data store. We've made this choice because, though the state database supported by Fabric which is

LevelDB or CouchDB supports file storage, IPFS is a public decentralized data store and brings added transparency to the entire procedure.

Identities in Composer are issued as business network cards. Users interact with the network using these business network cards. A single identity is stored in a business network card. Historian registry is used to record all the successful transactions, including the participants and identities that submitted them. REST server generates Open API for the business network and supports Create, Retrieve, Update and Delete (CRUD) operations.

As mentioned in the Proposed Designs section, our work involves implementation of the two designs we've developed. While the first design was implemented on Ethereum, the design 2 that was proposed was implemented on Composer.

The benefits of the second design's implementation as mentioned earlier are that, the entire verification process is initiated and completed by the candidate. Also, the smart contract written is such that the employer can verify the authenticity of the candidate's credentials and certifications even without having direct access to the certificate's contents. Now, we describe the entire procedure to see, how our Composer implementation caters to the second design proposed.

i. (Transaction 1 / Request): Candidates requests the certificate from the university by providing the necessary inputs such as his ID and the reason for the request. He does this by creating an asset called *request* which would contain the above-mentioned inputs a long with the requestID. This asset is sent to the university using *sendRequest* transaction. The request asset has two other attributes such as transaction ID and the submit status. These attributes cannot be controlled directly by any user and they get updated by the smart contract upon the submission of a request.

ii. (Transaction 2 / Response): University reviews the candidate's request and publishes the encrypted digital certificate on a decentralized data store. This encryption is done using the public key of the candidate, which is available to the university either at the time of candidate enrolment or through interaction on the smart contract. The decentralized data store returns the hash of the encrypted certificate. The university creates a *response* asset that contains the hash of the published certificate as returned by the decentralized data store. This asset contains a field called responseID with which this response can be identified when queried. Also, as a part of this asset, the university would specify the name of the degree or diploma awarded and the class obtained. This asset is then sent by the university to the candidate in a *sendResponse* transaction. This *sendResponse* transaction contains a reference to the request to which the university is responding and another reference to the response generated. The response asset also has two other attributes such as transaction ID and the submit status. These attributes cannot be controlled by any user and they get updated by the smart contract upon the submission of a response.

iii. Candidate decrypts the certificate using his private key, after downloading a copy of the encrypted certificate from the decentralized data store.

iv. (Transaction 3 / Validation): Candidate creates an asset called *proof* that would contain the requestID of the request sent by the candidate to the university and the responseID of the response sent from the university to the candidate. Also, the proof asset would contain the hash of the encrypted certificate which the candidate mentions. This proof contains a field called check hash status which cannot be set by the candidate. In addition to this the proof also contains two other fields, namely the transaction ID and the submit status. These attributes cannot be controlled by any user. They get updated by the smart contract after the submission of the *sendProof* transaction. Once the candidate submits this asset called proof as a part of the send proof transaction, the underlying smart contract checks the hash of the certificate as mentioned by the candidate against the hash of the certificate as mentioned in the response from the university. Check hash status is a Boolean field which would get set based on the result of this verification. Since this transaction is addressed to the employer, the employer can view the result of the verification. Also, after this transaction, the employer would have the permissions to view the request, response and the proof. This way, the employer can obtain the validation of the certificates of the candidate, without even viewing them.

There is no need for another transaction between the employer and the candidate, as the result of the verification process is transparent to both these entities. The advantage of this approach is that, there is no need for the university to explicitly be a part of the verification process. The credentials of the candidate can be viewed by the employer, who gains read access to the response asset after the send proof transaction. There is no need for the employer to actually see the contents of the certificate because the verification is done through a smart contract which all the parties trust. Also, the employer has access to the response asset which contains the credentials and the degree or diploma awarded to the candidate. Therefore, as a part of this second design, sharing the contents of the certificate between the candidate and the employer is made optional. We say, the contents of the certificate because, though the employer has access to the hash of the certificate on the decentralized data store, he or she doesn't have access to the contents because the certificate is encrypted using the public key of the candidate. Another salient feature of this verification procedure is that, the entire process of certificate and credential verification is initiated and completed by the candidate.

This four-step verification procedure is what is used in the implementation on Hyperledger Composer. As we can see from the steps mentioned, this way of verification eliminates the direct interaction between employer and university. This is a great benefit because, it makes the entire verification process driven by the candidate directly and is a completely transparent process. In addition to this, there is no need for the employer to create any asset or submit any transaction.

Also, the entire verification process is logged with the help of transactions recorded on the blockchain.

We now elaborately describe all the rules mentioned in the access control list.

1. Candidate:
    a. Has read access to his or her details and not the details of other candidates.
    b. Has read access to the details of all employers.
    c. Has read access to the details of all the universities.
    d. Has read access to only the requests created by him.
    e. Can create requests.
    f. Can delete the request created by him only if the request hasn't been submitted.
    g. Can update the request created by him only if it is not submitted earlier and only during a *sendRequest* transaction.
    h. Candidate can only send requests created by him and only to the university where he has enrolled himself. This is done using the *sendRequest* transaction. Also, a candidate can't resend a request.
    i. Candidate can only read the responses sent by the university directly to him.
    j. Candidate can create his proofs and delete those proofs that aren't submitted. Also, the proof must contain references to request and response to which the candidate has access to. These requests and responses must have been submitted by the time of creation of the proof.
    k. Candidate can update the proof generated only if the proof hasn't been submitted earlier. Also, he can do this only through the *sendProof* transaction.
    l. Candidate has read access only to the proofs created by him. He can submit the proofs using the *sendProof* transaction, only if the proof hasn't been submitted earlier.
2. University:
    a. Has read access to the details of all the employers.
    b. Has read access to the details of all the universities.
    c. Can enrol and read the details of candidates belonging to the same university.
    d. Can only read the requests meant and submitted to it.
    e. Has read access to all the responses generated by the university.
    f. Can create and update responses that aren't submitted earlier.
    g. Can update the responses generated by the same university and are not submitted earlier. Updating the response can be done only during the *sendResponse* transaction.
    h. Can send the response created by itself.
3. Employer:
    a. Has read access to the details of all other employers.
    b. Has read access to all the details of all the universities.
    c. Has read access to the details of all the candidates.
    d. Has read access to the proofs sent that are meant and submitted to him.

All the users have the permissions to read the request, response and the proof transactions. One must note that this gives access permissions to view the transaction and not the actual asset itself.

C. Results

This section focuses on the results obtained after the implementation of the two designs on different platforms. As mentioned earlier, the justification for the choice of platforms will be mentioned in this section. Also, we'll be discussing the performance in terms of time taken per transaction on either of the platforms.

We've chosen Ethereum as the platform for the first implementation because it is a public permission less blockchain. For the first design that we proposed, a public blockchain would be a good choice of implementation because there are no permissions involved in this implementation. The entire implementation procedure is based on encryption and decryption at every stage. So, for such an implementation, having a public blockchain would be better because all the nodes are public, data is encrypted and the transaction details are transparent.

The smart contract used for our decentralized application has been written in Solidity programming language, which is a standard program for writing smart contracts. This smart contract has been deployed on the Ropsten Testnet of the Ethereum network. The details of the deployment of the contract are given below:

- Transaction hash: 0x3e9b31f4841ce7a4a46e0a2bda6b22dad0c7a63886e3909e53ee35c0cd806db7
- Block number: 5254155
- Transaction fee: 0.001955713 Ether
- Gas limit: 1,955,713
- Gas used: 1,955,713 (100%)
- Gas price: 0.000000001 Ether
- Confirmation time: 43 seconds

Using the above-mentioned transaction hash, one can view the transaction details on ropsten.etherscan.io on the Ropsten Testnet of the Ethereum network. Since we've used a Testnet for deployment purposes, the ether used was fake ether. If we have to give a real currency value to the cost of deployment of the contract, it is $0.2705 or 18.70 Indian rupees as of 6:30 pm on 22nd March 2019.

In the following table we've logged the transaction details of 10 request transactions submitted on the Ropsten Testnet. We've done this to provide a comparison of performance of our application on Ethereum and Fabric platforms. Any transaction can be used for this comparison and request transaction was used because it is the first transaction that would take place during the verification process.

Table 1: Results of smart contract execution on Ropsten Testnet

Transaction hash	Block number	Transaction fee (ether)	Gas limit	Gas used	Gas price (Gwei)	Confirmation Time (seconds)
0x2a6118f5dfc4c15867dc2339f6fb996a 4abd87ff224f4bb77441609cdb8d89a5	5254166	0.000152115	228172	152,115	1	21
0xd4cb3a6bde1dd69f143d38102865c58 4757f5db130e06dde2f31817af08315ba	5254204	0.000062115	93172	62115	1	32
0xab0bcf00ac3cd0d2e1a25024dea7b7f4 881c39401d0447986de04296ab2603a1	5254216	0.0012360885	93172	62115	19.9	68
0x078963f3e4469332ac3b707a5a580b1 8310ed8f4b7dec3e84b166a593e60460f	5254228	0.0012423	93172	62115	20	15
0x60fdf7f47ef147b526fecfd5943b86d25 2274bdd7eca687674603832796ce1c0	5254245	0.00012423	93172	62115	2	72
0x7cedcef7b94de8c2e3fcc4f0dce43fee1 4c9aa6159b24f29b37e116f55162143	5254255	0.000434805	93172	62115	7	63
0x1ec149140ab7ec4fdf0d6aa1b3568319 ffefdcc570c1ab6045998b2f25b4cdb2	5254266	0.000434805	93172	62115	7	33
0x2a6429826aaa30f3e534246e5776882 71e8c25d8e898bd297688bf3d65ae4fa2	5254269	0.00012423	93172	62115	2	43
0xdae119e01f05beda078f9e3f343b856f c161296c07de1415da0f347202d15bc1	5254283	0.0012360885	93172	62115	19.9	19
0x80f822b43f2aaf6ec6fd923aba4034e8 ee0eacd015c0d4ff0991250f56a745f3	5254302	0.37269	93172	62115	6000	16

1Gwei = 0.000000001 ether.

We can use the transaction hash to know the details of the transaction on ropsten.etherscan.io. We've varied the gas price for the transactions and observed changes in the taken to confirm a transaction on the Ropsten Testnet. We observe that the *average confirmation time* for a transaction on Ropsten Testnet for our request transaction in these iterations is ***38.2 seconds***. Also, the *average transaction fee* paid is: ***0.0377737 ether***. This when converted to real currency translates to $5.2357 which is 361.85 Indian rupees as of 6:30pm on 22nd of March 2019.

The implementation of the **second design** that was proposed was performed on Hyperledger Composer. Composer was chosen as the platform for implementation because, the second design requires permissions set for users. Hyperledger Fabric, being the most popular platform for deployment of permissioned blockchains, we've made the choice to implement the second design on Composer.

As mentioned earlier the smart contract written on composer is a combination of 4 files namely, the Model file (.cto), the Access Control file (.acl), the Transaction functions file (.js) and the Query file (.qry). These files together form the business network archive (.bna) file. This file can be deployed on any Hyperledger Fabric network.

Being a private-permissioned blockchain, the transaction details are stored in a local historian registry that is accessible only to the participants that are a part of the network. So there is no publicly available log to see the transaction details as we could do in the case of Ethereum with the help of website by name ropsten.etherscan.io.

The time taken for the **deployment of the smart contract** on Composer is: *4 seconds*.

We've used the request transaction as a benchmark to compute the transaction confirmation time of Composer. Recall that we've used the request transaction as a benchmark for performance on Ethereum implementation also. Being a private-permissioned blockchain, there is no transaction fee for confirmation of transactions on Hyperledger Fabric (Composer). The average time taken for the **confirmation of a request transaction** on Composer has been observed to be: *0.86 seconds.*

In the Conclusion chapter, we provide some interpretation of the observed results of each of the implementations and compare the two platforms in terms of performance and efficiency.

VI. Conclusion

In this work, we've tried to address the problem of certificate and credential verification between the candidates, employers and universities. The existing methodology isn't transparent and involves placing trust on third parties. Due to lack of transparency in the existing procedure, fraudulent verifications can take place which can prove risky for the employer and the candidate. Also, the existing process is time consuming and brings additional overhead to the university and the employer.

In an attempt to address these existing issues, we've proposed two designs to complete this verification process in a completely transparent and decentralized manner while also preserving the privacy of the entities involved in the verification. We've achieved this with the help of blockchain technology. Blockchain technology helps in bringing decentralization, transparency, immutability of data and preserves user and data privacy. We note that, these are the exact properties lacking in the existing methodology of certificate verification.

Both the designs that we've proposed shift the responsibility onto the candidate to initiate and complete the verification procedure. This helps in reducing significant overhead from the university and the employer. We've implemented these designs, one on Ethereum and another on Hyperledger Composer. The reason why we chose these platforms for each design has been explained in the Results subsection of the Implementations section.

Upon implementing our design on each of these platforms we've noticed that there was a significant difference in the speed of transaction processing on each of these platforms. On Ethereum we've seen that a user needs to wait for an average request transaction confirmation time of 38.2 seconds. On the other hand, average request transaction confirmation time on Hyperledger Composer is just 0.86 seconds.

Based on the above observation we must not conclude that Hyperledger Composer is a better platform for development of decentralized applications when compared to Ethereum. This is because, Ethereum by design is a public permission less blockchain. So, the number of nodes involved is huge at any given point of time. Also, Ethereum implements Proof-of-Work consensus mechanism. Owing to the computational intensity involved in this consensus mechanism, it takes a

significant amount of time for the confirmation of a transaction.

On the other hand, Hyperledger Composer is a private-permissioned blockchain platform for development of decentralized applications. Due to this, the number of nodes participating in the network are significantly low compared to those present in a public blockchain. Also, the consensus mechanism involved is not as computationally intensive as the Proof-of-Work consensus used in Ethereum.

Another comparison of the results obtained on each of these platforms is relating to the cost of implementation. On Ethereum we had to pay 0.001955713 ether for deployment of a contract. There is no need for any fee for the deployment of contract in Composer. Also, the average transaction fee that we've paid on Ethereum is 0.0377737 ether. On Composer there is no need for any transaction fee. Again, this doesn't put the Hyperledger Composer platform on a higher pedestal compared to Ethereum. This is because, Ethereum being a public blockchain platform needs to incentivize miners who form the blocks on the blockchain. Also, having transaction fee on a public blockchain helps prevent the possibility of Denial of Service or Distributed Denial of Service attacks.

Because of the need for permissions in the design2 proposed, we use the Hyperledger Composer implementation. This design also proves to be more efficient because it eliminates the need for any transactions to be submitted by the employer. Also, there is no need for the employer to view the contents of the certificate. This is because the smart contract takes care of the verification procedure internally.

In the first design, the requirement for employer participation is only in the final stage of verification. The employer needs to verify the hashes of the certificates in the final stage and notify the candidate about the result of verification.

We can conclude that, though the transaction latency in either of the platforms is staggeringly different, the choice of each platform is justified given the advantages of transparency provided by public blockchains such as Ethereum and the ease of defining permissions provided by Hyperledger Composer. Also, compared to the average time of 30 days taken per verification we've brought the verification speed to near real time.

ACKNOWLEDGMENT

Our work is dedicated to Bhagawan Sri Sathya Sai Baba, Revered Founder Chancellor of Sri Sathya Sai Institute of Higher Learning. We thank Maestro Technology, New Jersey, USA for support. We acknowledge the encouragement from Mr. Adarsh Saraf, Software Engineer – IBM Research, Blockchain and Smart Contracts Lab, Bengaluru, India.

REFERENCES

[1] Cloudfare, "Famous DDoS Attacks | The Largest DDoS Attacks Of All Time," 2019. [Online]. Available: https://www.cloudflare.com/learning/ddos/famous-ddos-attacks/.

[2] M. Noor, "Is blockchain the new database?," May 2019. [Online]. Available: https://www.quora.com/Is-blockchain-the-new-database.

[3] S. Nakamoto, *Bitcoin: A Peer-to-Peer Electronic Cash System*, 2008.

[4] J. B. E. F. A. M. S. G. Arvind Narayanan, "Bitcoin and Cryptocurrency Technologies," 2016, p. 33.

[5] J. Benet, "IPFS - Content Addressed, Versioned, P2P File System," IPFS.

[6] M. Swan, Blockchain: Blueprint for a New Economy, O'Reilly, 2015.

[7] D. G. Wood, "ETHEREUM: A SECURE DECENTRALISED GENERALISED TRANSACTION LEDGER EIP-150 REVISION," 2014.

[8] Hyperledger, "An Introduction to Hyperledger," 2018.

[9] I. N. J. M. D. M. Kirill Kuvshinov, "Disciplina: Blockchain for Education," 2018.

[10] M. M. L. L. Initiative, "Blockcerts—An Open Infrastructure for Academic Credentials on the Blockchain," 24 October 2016. [Online]. Available: https://medium.com/mit-media-lab/blockcerts-an-open-infrastructure-for-academic-credentials-on-the-blockchain-899a6b880b2f.

[11] A. Murali, "Tamper-proof degree certificates to be India government's first blockchain project," 6 February 2018. [Online]. Available: https://factordaily.com/degree-certificates-india-blockchain-project/.

[12] Hyperledger, "Hyperledger Composer," 2019. [Online]. Available: https://www.hyperledger.org/projects/composer.