Privacy Protected Blockchain Based Architecture and Implementation for Sharing of Students' Credentials

Raaj Anand Mishra, Anshuman Kalla, An Braeken, Madhusanka Liyanage

Abstract

Sharing of students' credentials is a necessary and integral process of an education ecosystem that comprises various stakeholders like students, schools, companies, professors and the governmental authorities. As of today, all these stakeholders have to put-in an enormous amount of efforts to ensure the authenticity and privacy of students' credentials. Despite these efforts, the process of sharing students' credentials is complex, error-prone and not completely secure. Our aim is to leverage blockchain technology to mitigate the existing security-related issues concerning the sharing of students' credentials. Thus, the paper proposes a tamper-proof, immutable, authentic, non-repudiable, privacy protected and easy to share blockchain-based architecture for secured sharing of students' credentials. To increase the scalability, the proposed system uses a secure off-chain storage mechanism. The performance and viability of the proposed architecture is analyzed by using an Ethereum based prototypical implementation. The test results imply that requests can be executed within few seconds (without block-time) and the system has stability to process up to 1000 simultaneous requests.

Index Terms

Blockchain, Smart Contracts, Transcript, Data Sharing, Security, Privacy, Ethereum

Raaj Anand Mishra is with Dell EMC, Bangalore, India. e-mail: raaj_mishra@dell.com

Anshuman Kalla is is with the Centre for Wireless Communications, University of Oulu, Finland. email: anshuman.kalla@oulu.fi and anshuman.kalla@ieee.org

An Braeken is with Industrial Engineering Department (INDI), Vrije Universiteit Brussel, Brussel, Belgium. e-mail:an.braeken@vub.be

Madhusanka Liyanage is with School of Computer Science, University College Dublin, Ireland and the Center for Wireless Communications, University of Oulu, Finland. e-mail:madhusanka@ucd.ie and madhusanka.liyanage@oulu.fi

I. Introduction

Today, online sharing of documents is preferred because of the obvious advantages like fast (almost immediate), easy (requires minimum efforts), globally accessible, anytime available, etc. Nevertheless, the online sharing cannot always guarantee security, integrity, authenticity, confidentiality, provenance and availability of the shared files. Such guarantees are difficult, even in the presence of a dedicated facility like password-based trusted server since various attacks could be mounted [1]. In recent years, blockchain has come-up as a promising technology for various types of information systems where security, privacy and trust are of paramount importance [2]. Some of the latest areas where the use of blockchain has been very promising other than the cryptocurrency are supply chain management [3], smart cities [4], healthcare [5], Connected Autonomous Vehicles (CAVs) [6], fog/edge computing [7], public auditing of important services like cloud storage [8], smart grids [9], fake media [10], and Industry 4.0 using digital twin paradigm [11]. Yet another application area of blockchain that has recently attracted the attention of researchers is secured sharing of students' credentials among various stakeholders of education ecosystem [12], [13].

Blockchain can play an alleviating role by forming a hash-based chain of blocks where each block cryptographically seals a set of valid students' credentials[14]. Moreover, to off-load the blockchain and have a cost-effective solution, the proposed solution uses off-chain storage. In other words, students' credentials are stored (off-the-chain) on a distributed file storage system, and their digital fingerprints are put on the blockchain.

A. Motivation

Students' credentials like transcripts, letter of recommendation and all kinds of certificates (like diploma, degree, internship, training, migration, and character certificates) are important documents that stay with an individual for the entire lifetime. Secure sharing of these students' credentials is an integral part of both the education ecosystem and the recruitment process of companies. Every year several students add one (or more) credentials to their academic portfolio. All such credentials need to be carefully created, issued to students and relevant data must be preserved for future use by educational institutes, without exception. Many higher education institutions have their dedicated department for managing such a kind of academic information system that deals with students' credentials. As of now, to make the credentials legitimate and tamper-proof, institutes make use of numerous methods like assigning a unique number, putting a hologram, affixing a student's photograph, inscribing all the possible details of the students like date of birth, place of birth, parents' name, and registration/enrollment number, on the credentials itself. Over the years, the process has become quite complicated and time-consuming.

A similar level of complexities arises for all the entities, who are dealing with students' credentials, like companies, professors and other institutes. For instance, when a student applies for a job, the company carefully checks the authenticity of the received credentials. If required, the company may contact the host institution by phone-call or email to endorse the validity of the credentials. In spite of following such a tedious process, the overall system is insecure, and is facing difficulty to deal with tampered and fake credentials [15].

Further, students credentials are private information and when it comes to the issuing, viewing and sharing of such information, the privacy turns out to be of paramount importance. Specially, new privacy regulations (such as General Data Protection Regulation (GDPR)) legalize the requirement to protect the privacy of personal information [16]. Hence, the platform for students' credential should provide a sufficient level of privacy.

B. Contribution

To resolve the above challenges, the paper proposes the utilization of blockchain technology along with smart contracts. The main contributions of this paper are as follows:

- Introduce a novel blockchain-based pragmatic architecture for secure sharing of students' credentials among all the stakeholders in the education ecosystem.
- Propose a novel privacy projection mechanism to protect the privacy of the students' credentials.
- Utilize an off-chain storing mechanism to improve the scalability of the blockchain system.
- Develop a Decentralized Application (DApp) using Ethereum blockchain as a proof-of-concept of the proposed architecture.
- Conduct numerous tests to check the viability, compute costs, measure execution times and gauge the scalability of the developed prototypical DApp.
- Analyze the robustness of the developed DApp against the most widespread security attacks.

The remainder of the paper is organized as follows: Section II presents the related work and compares our work with them. Section III presents the proposed architecture and its core functionalities. The prototypical implementation is expounded in section IV. Next, section V elaborates the experimental results. Section VI analyzes the security features of the prototypical DApp. Finally, Section VII concludes the paper by drawing the future research directions. The demonstration video of the developed prototype can be found here [1].

[1]https://sites.google.com/view/secure-share2020/home

II. Related Work

Numerous advantages gained from the use of blockchain in education are immutability and provenance of uploaded credentials, peer-to-peer and secure interactions between stakeholders, transparency and trust within the system, and decentralized control with distributed digital ledger[17]. These advantages have impelled a variety of research works which are summarized in this section.

In [12], [13] and [18], authors discussed the possible use of blockchain in various education related applications such as management of school records, tracking of school assets, management of student privacy, parental opt-in/opt-out permissions, and distribution of public funds or private grants. In [19], authors presented the use of blocks to store assignments, awards, research works, etc. However, the paper does not propose any architecture or implementation strategy. The paper [20] proposed a mobile application to store and verify student certificates using Hyperledger (permissioned) blockchain. Records are stored encrypted in order to guarantee privacy. Authors compared previous existing works with their prototype in terms of on-chain storage and scalability.

Authors in [21] put forward a blockchain framework to enable the exchange of students registration data from school to learner to businesses (and vice versa). It makes use of a centralized database to extract learner data which makes the architecture partially decentralized. The paper does not cover implementation details as well as experimental analysis. The work [22] is a case-study on the BlockCert which is a blockchain platform to store students degrees and is implemented by Massachusetts Institute of Technology (MIT). Each stored credential has a unique URL which can be shared with others. Receivers of the shared URL can cross verify its authenticity by visiting the official website and passing that URL as an argument. Authors in [23] presented a blockchain-based architecture to store educational records on the blockchain. The work suggests that storing educational records on blockchain reduces the cost of storage as compared to that encountered with cloud storage. Privacy is obtained via the management of access control in the smart contracts and requires secure storage at the different database providers. However, the work does not carry any implementation nor any test-based result.

The work in [24] proposed a blockchain-based architecture that uses multi-signature to transfer academic credentials and course credits among stakeholders. The framework is built using Ark blockchain which is an open source and permissionless blockchain. The work seems to use on-chain storage which may lead to increase in cost and scalability issues as the number of users (and their data) increases. Moreover, the paper does not include implementation details and test results. Authors in [25] introduced $BcER^2$ as a blockchain-based repository for educational credentials. They used an open source framework called Hyperledger Composer to implement their architecture. The paper describes the architecture but

does not include any implementation details nor any test results. Author in [26] reported that Central New Mexico Community College uses blockchain to store student credentials so that students can get ownership of their academic credentials. Students can download the credentials on any device using their wallet address. Their proposal seems to use on-chain storage which is costly and has a scalability issue.

The paper [27] puts forth a blockchain-based prototype that consists of two types of users; admin and student. The admin can add a certificate to a student's account and the student can view it. The paper does not include costs associated with various smart contracts. The paper [28] presented a prototype to store and verify transcripts using blockchain. The prototype uses Neo4j as well as BigChainDB to store the transcript data and the metadata is stored on the Ethereum blockchain. The use of Neo4j might not allow the architecture to be completely decentralized and secure. Also, the work does not include cost computation of various smart contracts. A blockchain-based grade sharing system is proposed in [29]. However, it was not designed to share the data with external users. A very high-level analysis of providing students ownership of credentials with blockchain technology is presented in [26]. However, the proof-of-concept implementation is not included.

TABLE I: Comparison of our work with other pertinent works

References	**Type**	**Confidentiality**	**Authenticity**	**Integrity**	**Privacy**	**Off-chain Storage**	**Cost Computation**	**Test of Execution Time**	**Scalability**
Arenas et al., 2018 [20]	Permissioned	×	✓	✓	✓	×	×	×	×
Bessa et al., 2019 [25]		×	✓	✓	×	×	×	×	×
Sharples et al., 2016 [19]	Permissionless	×	✓	✓	×	×	×	×	×
Andreev et al., 2018 [21]		×	✓	✓	×	×	×	×	×
Young et al., 2018 [22]		×	✓	✓	×	×	×	×	×
Han et al., 2018 [23]		×	✓	✓	✓	×	×	×	×
Srivastava et al., 2018 [24]		×	✓	✓	×	×	×	×	×
Hope et al., 2019 [26]		×	✓	✓	×	×	×	×	×
Kanan et al., 2019 [27]		×	✓	✓	×	×	×	×	×
Arndt et al., 2020 [28]		×	✓	✓	×	✓	×	×	×
Turkanović et al., 2018 [30]		✓	✓	✓	×	×	×	×	×
Gräther et al., 2018 [31]		×	✓	✓	✓	✓ (Centralized)	×	×	×
Ocheja et al., 2019 [32]		✓ (Partially)	✓	✓	✓	✓ (Centralized)	✓	×	✓
Our Work		✓	✓	✓	✓	✓ (Distributed)	✓	✓	✓

Some of the existing works, which are closely related to our work are [30], [31] and [32]. The paper [30] proposed and implemented a system named EduCTX which can securely transfer the education credits between different institutes by avoiding language and administrative barriers. The authors work focused on (i) issuing of credits by an institute to the enrolled students, (ii) viewing of credits by the students, other institutes and organizations like companies for employment, and (iii) transferring of credits among institutes for promoting student exchange and educational mobility. However, the above three operations

are merely for academic credits and not for the entire set of academic credentials. Also, their architecture uses a consortium type of blockchain where any new institution is approved by the existing member(s) using a Delegated Proof-of-Stake (DPoS) algorithm. This approach may face a problem when none of the existing members can identify a new institute (trying to join) and thus no one is there to verify that new entrant. Further, when a new institution joins EduCTX system, it gets a reimbursement request from an existing member institution (who is about to approve the new entrant). This request is sent over a private link. If this link is not secure, an attacker may tap it and can play around. Further, at the time of registering a student, an institution creates a new account that involves a generation of public private key pair for that student. The institution sends the newly created public and private key pair to the concerned student using a private channel which might not be secure. Privacy is obtained by defining anonymous certifiers. However, the security of these certifications relies on the private key of one single authority.

The architecture proposed in [31] for student certificate management, uses a distributed file system for storing profile information of the certificate authority. However, it seems the actual certificates (in plaintext) are stored in a centralized system. If an attacker succeeds in getting access to this document management system, then all the sensitive private information can be leaked. Another issue is each time a student wants to share his/her credentials with any organization or a company, the request is approved by the institute managing the centralized system. This could be a cumbersome task as a student may send his/her credentials to several employers for which the admin needs to verify the credibility and then issue the access right.

In [32] authors proposed every institute to provide access via smart contracts to its data stores which is used for storing student's records and enables student's privacy. This architectural proposition leads to many security and privacy issues since the data stored are in plaintext and the databases are centralized. Moreover, even after an institution has given access rights to a concerned student, it is still possible that the institute may grant access to his/her learning logs to others. Thus, students do not have full control over their private information. Further, the work does not talk about how an entry of a new user (learner or provider) is authorized. Authors also proposed to link multiple accounts (i.e. one per institution) of a student by using one unique ID. However, discussion on how this unique ID is created, who approves and assigns this ID to a particular user seems to be missing.

Table I distinguishes our work with the existing related works. When it comes to secure issuing, viewing and sharing of all sort of academic credentials, then the major concern is the size of data. Using on-chain data storage will incur exorbitant costs that the stakeholder might not be able to bear, especially the students. Thus, the architecture proposed in the present work uses off-chain distributed file storage which improves scalability and reduces the operational cost of the system. Moreover, the present work

involves a specialized government body to issue a unique ID for every user. With this unique ID, a single life-long account is created for all the users. Unlike related works, our proposed work provides security features like confidentiality, authenticity, integrity and availability. Once a credential is issued, the student gets the full ownership of the credential. Most importantly, our work reveals the implementation details of the developed decentralized application and carries out numerous experiments to check the viability.

III. The Proposed Architecture

This section aims to present a secure and privacy protected architecture for the sharing of students' credentials. The overall architecture, ensuring both security and privacy, is proposed in two different phases. In the first phase, an architecture that is secure, trustful and scalable is presented. Security and trust are established with the use of blockchain technology, whereas, to improve the scalability off-chain storage is used. The second phase improvises the architecture by including privacy protection that avoids leakage of personal. The first phase is described in Subsection III-A, whereas, the second phase is discussed in Subsection III-B.

This work considers an education ecosystem that comprises of five different types of stakeholders. The distinctive roles played by each of these five different types of stakeholders are discussed as follows:

(i) Government Body: A specialized government body plays a vital role in creating a unique identity for every user. This unique identity is used to create a lifelong unique account for each user. A user can be of any type; student, school, professor or company (discussed next).

(ii) Schools: Any kind of educational institute like primary school, high school, college, or university is represented by the stakeholder 'school'. It plays a dual role both as an issuer and as a viewer. The reason being, a school issues credentials to the enrolled students when they fulfill the requirements, whereas, it views credentials of a new student, issued by previous school(s), at the time of admission.

(iii) Students: They need to view their credentials and securely preserve them in their academic portfolio for future use. Students have to provide access to their credentials when seeking admission in a new school or at the time of recruitment in a company. Thus, a student acts as a viewer as well as an access provider (to his/her credentials).

(iv) Companies: Predominantly, at the time of recruitment, the company's talent acquisition or human resource department needs to get access to view the valid credentials of an applicant. Alternatively, when a student undergoes a training or an internship at a company then that company has to issue a relevant credential. Thus like a school, a company can also be an issuer and a viewer.

(v) Professors: Like schools and companies, professors also play a dual role. At some times, they have to issue credentials (like internship or letter of recommendation) to their students. Whereas, for the

purpose of recruitment to some positions like research fellow, PhD, PostDoc, etc., they need to view the applicant's credentials.

Some of the notations used in this paper hereinafter are delineated in Table II.

TABLE II: List of Notations

Notation	**Description**
(d, Q)	Private and public key pair
M	Message or data in plaintext
M_e	Encrypted message or data
$\{M\}_d$	Digital signature of a message M using the private key d
$H(M)$	Hashing of M using some kind of hash function, for example SHA256
$E_Q(M)$	Encryption of M using a public key Q which results in an encrypted message M_e
$D_d(M_e)$	Decryption of an encrypted message M_e using a private key d which results in retrieval of M
ID_u	Unique identity of a user
$cert$	Digital certificate
$Credi$	Credential of a student
$Credi_e$	Encrypted credential of a student
$address$	Pointer to the location where a credential is stored on distributed file storage
OH	Original hash of a credential
TP	Temporary upload of a credential by a student

A. Architecture Without Privacy

A simple and pragmatic architecture for sharing students' credentials is shown in Figure 1. The proposed architecture comprises of the five different stakeholders, a blockchain infrastructure and a distributed file storage. All the interactions among the different stakeholders are being recorded on the blockchain in the form of immutable transactions. Smart contracts are deployed on the blockchain to manage the interactions and offer numerous functionalities.

The core functionalities of the proposed architecture are as follow:

(1) Registration of Users: This functionality enables a specialized government body to assign a unique identity ID_u to every user based on his/her legitimate request (cf. 1 in Figure 1). Once assigned, the ID_u remains with a user for a lifetime. The ID_u is digitally signed and stored on the blockchain by the government body. The exact way to create and assign unique IDs is beyond the scope of present work since it is a prerogative of the government body to formulate an appropriate policy to do so.

(2) Sign-up and Login of Users: After registration, each user needs to set-up a lifelong permanent account on the blockchain. This is done with the help of the *sign-up* functionality (cf. 2 in Figure 1). Based

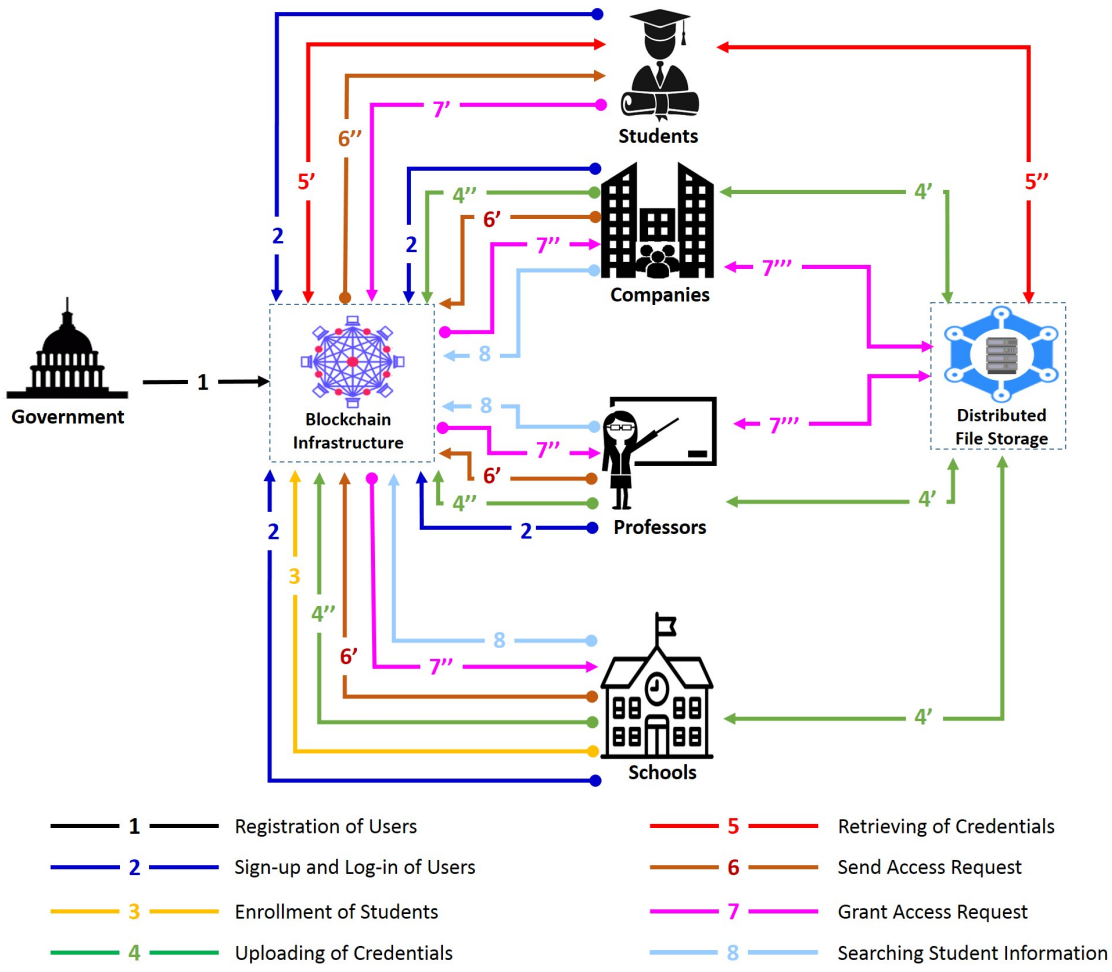


Fig. 1: Proposed architecture for sharing students' credentials (without privacy)

on the assigned ID_u, a user creates an account on the blockchain by providing the other mandatory information (like address, age, and type of stakeholder). This also leads to creating PKI-based login access (i.e. setting-up of private/public key pair). There are numerous ways to carry out the sign-up process. This work proposes a solution using Elliptic Curve Qu Vanstone (ECQV) certificates and is discussed in *Appendix B*. Using the ECQV mechanism for the sign-up process enables a user to generate a key pair based on the certificate issued by the government body. It has the advantage that only the user is aware of the private key and a unique relation can be made between ID_u and the corresponding public key. Moreover, ECQV enables any user to generate a public key of any other user X, provided ID_x and $cert_x$ (certificated of user X issued by a government body) is known.

(3) Enrollment of Students: This functionality is used by a school at the time of admission of a student (cf. 3 in Figure 1). Let, (d_{stud}, Q_{stud}) denotes the private and public key pair of a student. For enrolling

the student, a school looks up Q_{stud} on the blockchain to verify the existence of the associated account. On verification, the school enrolls the student by adding the public address, i.e. Q_{stud} to the list of enrolled students. This mapping is reflected in the student's account as well.

(4) Uploading of Credentials: The school issues credentials to its enrolled students at the end of an academic term or whenever students fulfill the requirements of a course or a program. Since these credentials (degree, certificate, transcript, etc.) are data of significant size, storing them on the blockchain becomes an expensive affair. With this view, the proposed architecture uses a distributed file storage (which is an off-chain storage) for storing the credentials. Uploading of a credential is done in three steps. *Step 1:* The school issues a credential by uploading it to the distributed file storage (cf. 4' in Figure 1). In return, the school gets an $address$ of the location where the credential got stored. *Step 2:* The school creates data $M = (address, metadata, Q_{stud})$ where metadata of a credential comprises of name and other information like date of issue, course and program. *Step 3:* Finally, it signs the message M and uploads the resultant (i.e. $\{M\}_{d_{school}}$) on the blockchain (cf. 4" in figure 1). Similarly, on completion of a training or an internship, a company (or a professor) uploads the completion certificate. This functionality is, thus, available to the three stakeholders; school, company and professor.

(5) Retrieving and Viewing of Credentials: The functionality is exclusively available to the students. It enables students to retrieve (and view) their credentials which are uploaded by stakeholders. To retrieve and view an uploaded credential, a student logs-in to his/her account and retrieves the $address$ of the new credential from the blockchain (cf. 5' in Figure 1). Using this $address$, the student downloads and views the credential (cf. 5" in figure 1).

(6) Sending Access Request: In order to view students' credentials, schools, professors and companies send access requests to ther concerned students (cf. 6' in Figure 1). Such a request Req along with the student's address Q_{stud} is signed by the sender using its private key d_{sender}. The digitally signed request $\{Req, Q_{stud}\}_{d_{sender}}$ is sent to the blockchain. The student retrieves the pending access requests from the blockchain and views them (cf. 6" in Figure 1). The legitimate requests are accepted and others are discarded.

(7) Granting Access Right: The acceptance of a request must be followed by the granting of access right to the requester. To do so, a student performs three steps. *Step 1:* creates data $M = (addresses, Q_{sender})$ with the addresses being the location pointers where the requested credentials are stored in the distributed file storage and Q_{sender} is the public key of the requester, *Step 2:* signs this message M, and *Step 3:* uploads the resultant i.e. $\{M\}_{d_{stud}}$ on the blockchain (cf. 7' in figure 1). The requester retrieves the $addresses$ of the credentials from the blockchain (cf. 7" in Figure 1). Finally, credentials

are downloads from the distributed file storage (cf. 7''' in Figure 1).

(8) Searching Student Information: When a student applies for an admission in a school or for a job in a company, the basic search functionality is made available to the relevant entity (i.e. school or professor or company). Students can decide which of their information will be visible to these stakeholders.

B. Architecture With Privacy Integration

Academic credentials are private data of students and when it comes to issuing, viewing and sharing of such data, privacy is of paramount importance. The above proposed architecture uses off-chain storage, which offers two distinct advantages; first, offloading blockchain thereby improving scalability and second, reducing the cost of transactions. However, the downside of using such kind of off-chain storage is that it becomes an easy point of attack and privacy can be compromised [33]. The idea of making the proposed architecture privacy protected implies that an attacker should not get access to any personal information and should not be able to derive any link between a student on the one hand and company, school or professor on the other hand. Thus, the intent is to enhance the security of the overall system by securing the off-chain storage as well such that students get complete ownership and access control of their credentials.

Figure 2 shows the blockchain-based architecture that proposes an effective mechanism to protect the privacy of students' credentials. This architecture modifies three of the above mentioned eight core functionalities. These are uploading of credentials, retrieving and viewing of credentials and granting access rights. Moreover, two new optional functionalities are also added which are shown with dotted lines in Figure 2. The three modified functionalities and the two new optional functionalities are discussed as follow:

(i) *Uploading of Credentials:* The process of uploading of credentials is divided into three steps. *Step 1:* When a new credential $Credi$ is issued to an enrolled student, the school computes the hash of the credential $OH = H(Credi)$. The resulting hash value (OH) is named as *Original Hash* and is used for ensuring integrity (explained in a while). *Step 2:* The school encrypts the credential using the public key of the corresponding student (i.e. $Credi_e = E_{Q_{stud}}(Credi)$). The encrypted credential ($Credi_e$) is uploaded on the distributed file storage which returns an $address$ (cf. 5' in Figure 2). *Step 3:* The school creates data $M = (address, OH, metadata, Q_{stud})$. Further, the school signs the message M and uploads the resultant (i.e. $\{M\}_{d_{school}}$) on the blockchain (cf. 5'' in Figure 2).

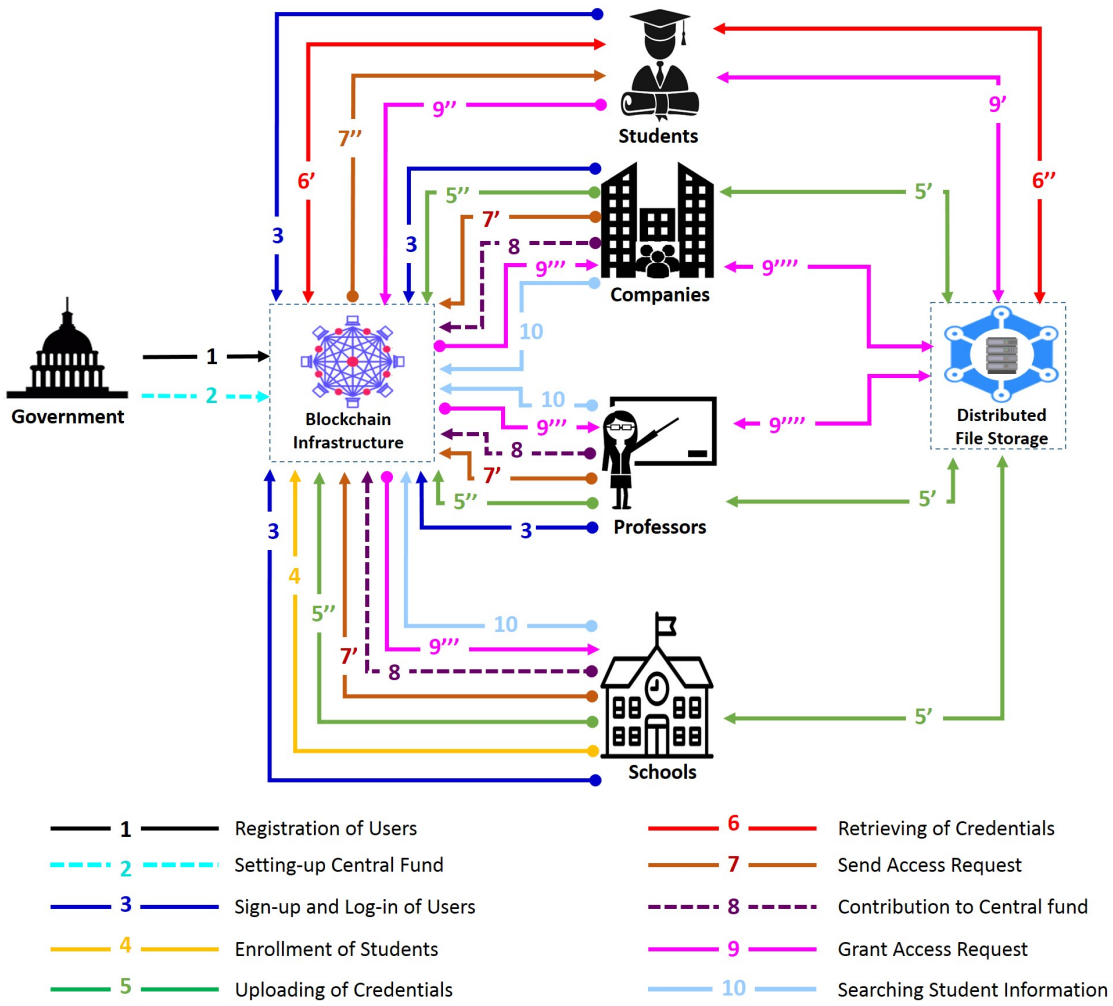


Fig. 2: Proposed architecture with privacy protection

(ii) *Retrieval and Viewing of Credentials:* To retrieve a credential, the student sends a request to the blockchain and gets the $address$ (cf. 6' in Figure 2). The student then downloads the encrypted credential from the distributed file storage (cf. 6" in Figure 2). S/he decrypts it using his/her private key i.e. $Credi = D_{d_{stud}}(Credi_e)$.

(iii) *Granting Access Right:* While a student grants access to his/her credentials, no one except the intended user(s) (i.e. a company, professor or a school), must be able to view the credentials. To do so, students are allowed to perform *temporary upload* of their credentials. The process is divided into five steps. *Step 1:* Student computes the hash of his/her credential $H_{TP} = H(Credi)$ and encrypts that credential using the public key of the intended user i.e. $Credi_e^{TP} = E_{Q_u}(Credi)$. *Step 2:* Student performs the *temporary upload* of $Credi_e^{TP}$ on the distributed file storage and

in return gets an $address$ (cf. 9' in Figure 2). Usually, a student is not supposed to upload a credential, only the school, company or professor can do it. But in order to ensure privacy, the student is permitted to make *temporary upload* of an already issued credential. *Step 3:* The student then creates $M = (address, H_{TP}, Q_u, metadata)$, signs it and the resultant $\{M\}_{d_{stud}}$ is put on the blockchain (cf. 9" in Figure 2). Here, one cannot ignore the possibility of a student trying to share a fake credential to get a job or an admission to some institute. To eliminate this possibility, the system compares the H_{TP} with the OH. Recall OH is the hash value of the same credential computed by the school at the time of issuing it. If the two are the same, it means that the integrity of the credential uploaded by the student during step *9'* is intact. Thus, the system accepts this operation, otherwise discards it. *Step 4:* The intended users get the access right and is able to retrieve the $address$ of the credential from the blockchain (cf. 9"' in Figure 2). *Step 5:* Using the $address$, the intended user downloads $Credi_e^{TP}$ from the distributed file storage, decrypts it using his/her private key i.e. $Credi^{TP} = D_{d_{stud}}(Credi_e^{TP})$ (cf. 9"" in Figure 2). The temporary upload can be automatically deleted by the system after an agreed or set duration of time thereby not hogging the storage space.

(iv) *Setting up of Central Fund:* It is interesting to note that temporary uploads by students incur some cost since it is a write operation on the blockchain. To compensate or subsidize the expenses at the students' end, a central fund exchange is established by the government body.

(v) *Contribution to Central Fund:* Depending on the policy agreements, stakeholders can be provisioned to contribute to the central fund. For instance, since companies want to recruit the best talents so they can be asked to contribute to the central fund before sending an access request to the potential candidates. The central fund raised in this manner can be used to reimburse students. Depending on the agreed policy reimbursement can be partially or fully, allowing students to avail subsidised or free access to the decentralized system.

IV. Implementation

This section expounds the implementation details of the developed prototype which is a Decentralized Application (DApp).

A. Background Technologies

Numerous approaches are possible to build a DApp given the fact that application development using blockchain is yet to mature. The approach used in this work makes use of the following technologies:

1) MetaMask[2]*:* It is a browser extension which provides features to interact with Ethereum. MetaMask permits a secure sign-in process, provides a user interface to manage different identities and enables digital signing of all the transactions taking place on the blockchain.

2) Web3.js[3]*:* It provides a collection of libraries to develop client side applications that interact with Ethereum. Web3.js interacts with MetaMask using the Remote Procedure Call protocol and connects it to the client application. Some of the important functions provided by these libraries are used to send Ether from one account to another, to deploy smart contracts, to read/write data from smart contracts, to call methods from deployed smart contracts, etc.

3) Next.js[4] *and React*[5]*:* Next.js is a React framework which is used to build composable web applications with functionalities like building a web application through modular programming, built-in routers to provide routing inside the application and also has the Hot Module Reload (HMR) which listens for changes made to the application and reflects the changes in real-time without restarting the server.

4) eciejs[6]*:* It is a javascript/typescript package which is used for key generation using elliptic curve cryptography.

5) Crypto[7]*:* It is a built in module in Node.js which provides cryptographic functionalities such as generating hashes, ciphers, deciphers, signing and verification of messages.

6) IPFS[8]*:* It is a distributed peer-to-peer file system for storing data. It works on the principle of content based addressing. When data is uploaded, a hash value of the data is returned which can later be used to retrieve the data. IPFS is used as a distributed file system in the implementation.

B. Overview

Three different types of stakeholders are considered for first-level implementation, which are school, student and company. As mentioned in Section III, a unique identity ID_u is allotted to every user by the government entity. For our implementation purpose, the Ethereum account ID provided by the MetaMask is used as ID_u. The ID is further mapped in such a way that it not just uniquely identifies a user but

[2]https://metamask.io/

[3]https://web3js.readthedocs.io/en/1.0/

[4]https://nextjs.org/

[5]https://reactjs.org/

[6]https://www.npmjs.com/package/eciesjs

[7]https://nodejs.org/api/crypto.html#crypto_crypto

[8]https://ipfs.io/

also represents the type of stakeholder, i.e. school, student or company. In the developed DApp, each type of stakeholder has a different dashboard and is enabled with a set of functionalities.

1) School Dashboard: It provides a school with an option to add students using their Ethereum address to the list of enrolled students. For the enrolled students, the student dashboard offers the option to upload their credentials. Before uploading, the school computes *original hash* OH. Moreover, the credential (to be uploaded) is encrypted using the public key of the corresponding student. The encrypted credential is then uploaded on IPFS which returns a hash value. This hash value is named as *IPFS hash*. The IPFS hash, the original hash and metadata of the credential along with the associated student ID are pushed to the Ethereum.

2) Student Dashboard: It enables the student with an option to view his/her credentials so far being uploaded by the school(s). To do so, the student retrieves the IPFS hash values of the credentials from Ethereum and then downloads the encrypted credentials from IPFS. The student decrypts the credentials using his/her private key. Another option permits a student to view the access requests received from the companies. An access request has a description of the credential(s), a company desires to view. Further, to allow a student to share and grant access of his/her credentials to the requesting company, another option is provided by the dashboard where a student performs a temporary upload of the credential(s) to be shared. Moreover, before uploading, the credential(s) is encrypted using the company's public key.

3) Company Dashboard: It facilitates a company with an option to view the list of schools and the students (enrolled under the respective schools). The viewing-list contains only the Ethereum account IDs and nothing else. Further, the company dashboard is equipped with an option to send an access request to any student to view his/her credential(s). Later, when the student grants the access request, the company can view the credential(s) after decrypting them using his/her private key.

Following are the assumptions for the current prototypical implementation:

- It is assumed that the mechanism that allows companies to publish current job vacancies is already in place.
- Moreover, it is considered that all the students apply to the published job vacancies and based on the company's selection criteria, scrutiny is carried out. Thus, the current implementation allows a company to directly select students and send access requests to them.

C. Application Stack

The application stack of the developed DApp consists of a front-end client which at the back-end runs on a decentralized server.

1) Front-End Client: Users interact with the DApp through a web browser installed with *MetaMask* extension which connects the application to the blockchain. The User Interface (UI) is built on *Next.js*. The DApp sends various kinds of requests to the decentralised back-end server using functions provided by *Web3.js* and *MetaMask*. For privacy integration, an external javascript library *eciejs* is used for key generation and encryption/decryption process. The front-end also uses *crypto* for generating the sha256 hash of the original credential.

2) Decentralized back-end server: Most of the requests generated by the front-end are handled at the back-end by the smart contracts deployed on the Ethereum. These requests are defined as functions in various smart contracts. Few requests are served by IPFS.

D. Various Smart Contracts

To develop a prototype, nine different smart contracts are designed based on the architecture proposed in Section III and are discussed below.

1) User Contract: The contract is employed for creating and storing the details of the signed-up user on the blockchain as well as retrieving them. The detailed structure (i.e. variables and functions as well as their description) of the user contract is shown in Figure 3. Various functionalities provided by the contract are: (1) to check if the user is already signed-up or not, (2) If not then it allows the user to sign-up by storing the required details on the blockchain and (3) If a user has already signed-up then (as and when required) it enables the user to retrieve the stored details from the blockchain. User contract corresponds to the core functionality named *sign-up and login of users* as mentioned in section III.

2) File Contract: The structure of File Contract is shown in Figure 4. The contract is executed when the details (metadata) of the credential (file) are to be uploaded or fetched by a user (on and from Ethereum). The details include title, description and hash value(s) generated by IPFS. This contract features three functions; (i) to upload the details of the file (already uploaded on IPFS) on the Ethereum blockchain, (ii) to get the count of the files under the ownership for a given Ethereum address, and (iii) to download the details of the file under the ownership for a given Ethereum address. File contract corresponds to two core functionalities which are *uploading of credentials* as well as *retrieving and viewing of credentials*.

3) School Contract: As exhibited in Figure 5, one of the functionalities this contract offers to a school is adding new students to the list of enrolled students. Moreover, exclusively for the enrolled students, this contract permits a school to upload the details of the credentials (already stored on IPFS) and when required retrieve the same. The school contract uses instances of file and student contracts as depicted in Figure 5. School contract allows to execute four core functionalities *sign-up and login of users, enrollment of students, uploading of credentials, searching student information* as discussed in Section III.

User Contract		
Variables		
Type	**Name**	**Description**
struct	User	To store name and designation of users
address [ ]	addressToUser	Public array containing user details mapped to their Ethereum addresses
boolean [ ]	userExists	Public array of Boolean values depicting if the Ethereum addresses are mapped to existing respective accounts
address [ ]	addressOfUsers	Public array to store users' Ethereum addresses
Functions		
Name	**Description**	
hasUser	To check if a given user exists with the true boolean value in the array userExists	
createUser	If user's Ethereum address is not mapped to an existing account then it stores values of name and designation (max 20 bytes) in User structure, thereby creating an account. Also, it adds that user's Ethereum address to addressOfUsers and updates userExists as well	
getUserAddress	Returns user's Ethereum address with existing accounts i.e. values from addressOfUsers array	
getUser	Returns name and designation stored in public array addressToUsers for a given user's Ethereum address	

Fig. 3: Structure comprising variable and functions of User Contract

4) Student Contract: Figure 6 depicts the structure of the student contract. This contract is used to retrieve the details of the credentials (files) and the count of total number of credentials under the student's ownership. The student contract uses an instance of the file contract. Student contract enables two core functionalities which are *sign-up and login of users* as well as *retrieving and viewing of credentials*.

5) Company Contract: When a user logs in as a company, this contract facilitates the user to fetch the list of schools and subsequently, the list of enrolled students for the selected school. Figure 7 reveals the structure of the company contract. As shown in this figure, the company contract uses an instance of the school contract. In terms of core functionalities, this contract provides two core functionalities which are *sign-up and login of users* as well as *searching student information*.

File Contract		
Variables		
Type	**Name**	**Description**
struct	File	To store IPFS hash, title, description and original hash of a file (credential)
File[ ]	ownerToFiles	Contains the mapping of the files to the students' Ethereum addresses
Functions		
Name	**Description**	
uploadFile	Stores the metadata of the file on the blockchain along with the associated student's Ethereum address	
getFileCount	Returns the total number of files under the ownership of a given student's Ethereum address	
getFile	Returns (retrieves) the metadata of the file under the ownership of given student's Ethereum address	

Fig. 4: Structure comprising variable and functions of File Contract

6) Request Contract: This contract is called when the logged-in user is either a company or a student. When the user is a company, this contract enables the user to send an access request to a student, to retrieve and view the credentials under the ownership of that student. Before forwarding the access request, the request contract validates the existence of the receiver's account. On the other hand, if the logged-in user is a student, this contract enables the user to view the access requests received by the company. The contract also keeps track of a number of access requests sent by a company and several access requests received by a student. Figure 8 presents the structure of the request contract. Request contract permits to execute two core functionalities which are *sending access request* and *granting access right*.

7) CertificateAuthority Contract: Initially, when a new user undergoes a sign-up process, this contract is used to store the generated public key of the user on the blockchain. Later, for ensuring privacy, this contract is used to fetch the receiver's public key and perform the necessary encryption. The structure of the CertificateAuthority contract is shown in Figure 9. This contract is used to implement three core functionalities; *sign-up and login of users*, *uploading of credentials* and *granting access right*.

8) ShareFile Contract: The main aim of this contract is to enable students to share their credential(s) to the requesting companies. This contract is called when the logged-in user is either a student or a company. When a user is a student, this contract provides the user with the functionality to upload the metadata of their credentials to be shared on the blockchain. On the other hand, if the logged-in user

School Contract * Inherits User Contract		
Variables		
Type	**Name**	**Description**
address [ ]	schoolStudents	Public array containing addresses of enrolled students
boolean	studentInSchool	Boolean value to check if a given student is an enrolled student or not
address [ ]	schoolStudent	Public array containing addresses of students enrolled under a particular school
FileContract	filecontract	Instance of deployed File Contract
StudentContract	studentcontract	Instance of deployed Student Contract
Functions		
Name	**Description**	
SchoolContract	Constructor that creates instances of File Contract and Student Contract using the filecontract address and studentcontract address respectively	
addStudent	Checks if there exists an account for given student' Ethereum address, further checks if the student is not enrolled in any other school and then enrolls the student by adding its address in schoolStudent array. Accordingly update studentSchool array as well as schoolStudent array (if required)	
hasStudent	To check if the student exists in the given school or not	
getStudent	Returns student's address enrolled under the given school's Ethereum address (no other detail is retrieved except the address)	
getStudents	Returns all the students' addresses enrolled under the given school's Ethereum address (no other detail is retrieved except the address)	
studentCount	Returns the total number of students enrolled under the given school's Ethereum address	
uploadFile	Checks if the student's account exists and if true, stores the metadata of the file on the blockchain along with the corresponding student's Ethereum address	
getFileCount	Returns the total number of files under the ownership of a given student's Ethereum address	
getFile	Returns the metadata of file under the ownership of given student's Ethereum address	

Fig. 5: Structure comprising variable and functions of School Contract

is a company then the contract allows the user to retrieve the total number of credentials being shared and retrieve the metadata of those credentials. The structure of this contract is shown in Figure 10. The contract implements one core functionality which is *granting access right*.

9) FundRaising Contract: The idea behind creating this contract is to facilitate students to use DApp without bearing financial charges. Rather, the school and companies must contribute to a central fund

Student Contract * Inherits User Contract		
Variables		
Type	**Name**	**Description**
FileContract	filecontract	Instance of deployed File Contract
Functions		
Name	**Description**	
StudentContract	Constructor that creates an instance of File Contract using the *filecontract* address	
getFileCount	Returns the total number of files under the ownership of a given student's Ethereum address	
getFile	Returns the metadata of file under the ownership of given student's Ethereum address	

Fig. 6: Structure comprising variable and functions of Student Contract

Company Contract * Inherits User Contract		
Variables		
Type	**Name**	**Description**
SchoolContract	schoolcontract	Instance of deployed School Contract
Functions		
Name	**Description**	
CompanyContract	Constructor that creates instance of School Contract using the schoolcontract address	
getSchoolAddresses	Returns only the Ethereum addresses of the schools stored on the blockchain (no other detail is retrieved except the address)	
getSchoolStudentAddresses	For a given school's Ethereum address, it returns the Ethereum addresses of students enrolled in that school (no other detail is retrieved except the address)	

Fig. 7: Structure comprising variable and functions of Company Contract

such that the expenses incurred at the student's end are compensated. It is proposed that a specialized government body deploys this smart contract using its own Ethereum account and by doing so that the government body becomes the owner of this contract. Based on the policy agreements, stakeholders like schools and companies can be provisioned to contribute to the owner's account from their account using this contract. The central funds is to be used to meet-out students' expenses by transferring the required

Request Contract		
Variables		
Type	**Name**	**Description**
struct	Request	To store description of a request as well as sender's and receiver's Ethereum addresses
Request [ ]	sentRequests	Public array used to map sent requests to senders' Ethereum addresses
Request [ ]	receivedRequests	Public array used to map received requests to students' Ethereum addresses
Functions		
Name	**Description**	
sendRequest	Checks the validity of given destination's Ethereum address, verifies the length of the description and finally sends the request to the destination	
getSentRequestCount	Returns the total number of requests sent by the given sender's Ethereum address	
getReceivedRequestCount	Returns the total number of requests received by the given receiver's Ethereum address	
getReceivedRequest	Retrieves the description of the request from the blockchain for a given student's Ethereum address	
getSentRequest	Retrieves the description of the request from the blockchain for a given sender's Ethereum address (used as to display the requests sent by the sender to itself)	

Fig. 8: Structure comprising variable and functions of Request Contract

amount, as and when required, to the student's Ethereum account. This ensures students to have free or subsidised access to the DApp. For example, when a company intends to send an access request to a student (i.e. job applicant) for viewing his/her credentials, the company first needs to contribute the amount required by that student for sharing the requested credentials. Of course, the amount required will depend on the number of credentials requested by a company from a student. Figure 11 depicts the structure of the FundRaising contract.

E. Interaction Between Various Contracts

The implementation invokes above contracts based on the designation of the user and types of operations being performed, at times concomitantly, as depicted in Figure 12. *User* contract stores the details of a user such as name, designation with the account's Ethereum address. This contract is inherited by *school*, *student* and *company* contracts to effectuate sign-in and login functionalities. The *school* contract makes

CertificateAuthority Contract		
Variables		
Type	**Name**	**Description**
address	publicKeys	Public array that maps the user's address to the respective public key
Functions		
Name	**Description**	
setPublicKey	Stores the user's public key in the publicKeys array	
getPublicKey	Returns the public key for a given user's address	

Fig. 9: Structure comprising variable and functions of Certificate Authority Contract

ShareFile Contract		
Variables		
Type	**Name**	**Description**
struct	File	To store the IPFS hash, original hash, title and description of the file
File [ ]	sharedFiles	Contains the mapping of the shared files to both the company and student's ethereum address
Functions		
Name	**Description**	
uploadFile	Stores the metadata (original hash, IPFS hash, name, description) of the shared file on blockchain along company's Ethereum address	
getFileCount	Returns the total number of shared files under the ownership of a given company's Ethereum address	
getFile	Returns (retrieves) the shared file under the ownership of a given company's Ethereum address	

Fig. 10: Structure comprising variable and functions of ShareFiles Contract

use of an instance of *student* contract and an instance of *file* contract. It uses the *student* contract to check if a student is already enrolled or not. Whereas, it makes use of the *file* contract to upload the metadata of credentials of the enrolled students. Once the metadata of the credentials is uploaded, the *student* contract also makes use of an instance of *file* contract to retrieve the same. Further, *company* contract makes use of an instance of *school* contract in order to fetch the list of schools and the list of

FundRaising Contract		
Variables		
Type	**Name**	**Description**
address	contributions	Public array that maps the user's address to the respective amount they contribute
uint	totalContributors	Unsigned Integer value that represents the total contributors
uint	minimumContribution	Unsigned Integer value that represents the minimum contribution required to contribute
uint	raisedAmount	Unsigned Integer value that represents the total amount raised by contributions
address	admin	Represents the Ethereum address of the administrator of the contract
Functions		
Name	**Description**	
contribute	Function allows the user to contribute to the fund if the contribution is greater than the minimum contribution specified	
makePayment	Function that allows only the admin to transfer required amount of ether to a student's account	

Fig. 11: Structure comprising variable and functions of Fund Raising Contract

enrolled students under a given school. This helps a company to select the students for the enrollment list and then send access requests to those selected students.

To send an access request, an explicit call is made to *request* contract by a company user from the DApp. In other words, *request* contract is being called by a company which means that the *company* contract has already been called by the logged-in user. To view the access requests received, the student user also makes an explicit call to the *request* contract. In essence, both company and student use *request* contract. After viewing the access request if the student (user) decides to share the requested credentials, an explicit call is made to *shareFile* contract from the DApp. Thus the *shareFile* contract enables a student to perform a temporary upload of the metadata of the credentials to be shared on the blockchain. The company also makes an explicit call to *shareFile* contract to retrieve the metadata of the shared credentials.

CertificateAuthority contract can be explicitly called by any user whosoever wants to fetch a public key of a given user's Ethereum address. Company and school users will use *FundRaising* contract to contribute to the central fund which will be later used to transfer the required amount to students' account to let students go without paying.

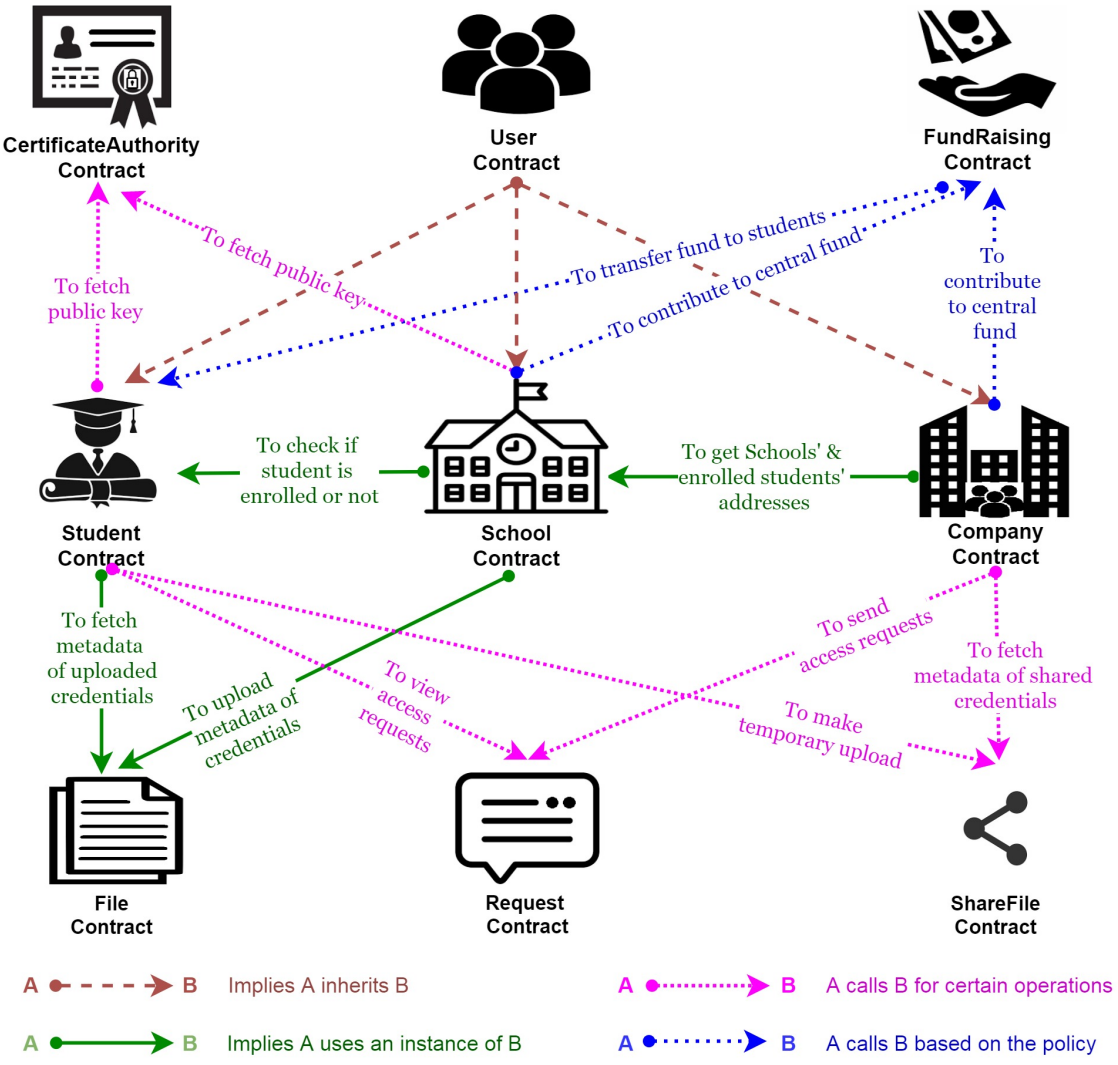


Fig. 12: Interaction between various smart contracts

V. Experiment Results and Discussion

The viability and performance of the developed DApp are evaluated by performing different tests using the experimental setup shown in Figure 13. To carry out the evaluation, Rinkeby Test Network[9] is used. Rinkeby is one of the public test networks provided by Ethereum. It uses Proof-of-Authority (PoA) consensus algorithm. Ether, the cryptocurrency token, used in Rinkeby has no monetary value. Infura API is used to communicate with Rinkeby network.

[9]https://www.rinkeby.io/#stats

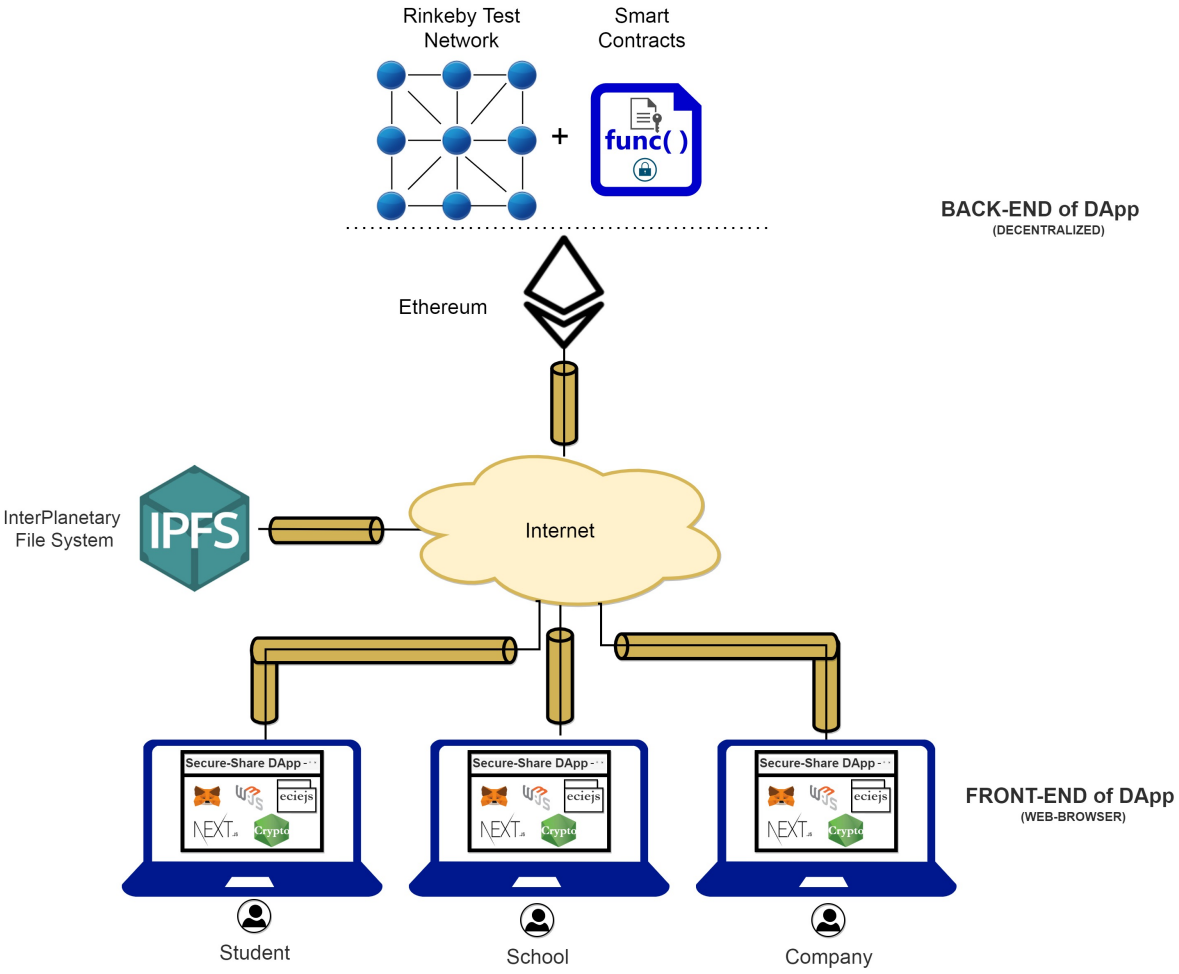


Fig. 13: Experimental setup for running various tests

A. Computation of Cost for Smart Contracts and Important Functions

To securely manage and ensure availability of the underlying decentralized P2P network, the blockchain must give incentives to miners. Thus, miners are rewarded for validating transactions and mining new blocks. In Ethereum, the reward is generally based on the computational power required to validate a transaction. These rewards are paid by the sender (of the transaction) in terms of gas to the winning miners. Hence, to ensure the economic viability of the DApp, it is important to carry out cost computations for the smart contracts and the important functions.

A comparative view of the gas required for the deployment of various smart contracts without the privacy and with privacy integration is presented in Table III. Two types of costs are encountered during the deployment of a smart contract on the Ethereum blockchain; transaction cost and execution cost. The transaction cost is the gas consumed when a smart contract is sent for validation along with necessary

data. The execution cost is the gas consumed for executing a smart contract and it depends on the number of variables used, the number of operations performed and the number of function calls made. Remix[10] is used to calculate values of both costs. Table III evinces that the transaction and the execution costs

TABLE III: Deployment cost (Gas) of Smart Contracts without and with privacy

Smart Contracts	Before Privacy				After Privacy			
	Transaction Cost	**Execution Cost**	**Total Cost**	**Gas Limit**	**Transaction Cost**	**Execution Cost**	**Total Cost**	**Gas Limit**
User Contract	268268	671302	939570	3000000	268268	671302	939570	3000000
File Contract	303532	812244	1115776	3000000	347164	942781	1289945	3000000
Student Contract	372792	1014987	1387779	3000000	384752	1053027	1437779	3000000
School Contract	588300	1680974	2269274	4000000	612432	1754851	2367283	4000000
Request Contract	353420	919555	1272975	4000000	353420	919555	1272975	4000000
ShareFile Contract	234396	556281	790677	4000000	386468	1077720	1464188	3000000
Company Contract	338408	894342	1232750	4000000	338408	894342	1232750	4000000
CertificateAuthority Contract	-	-	-	-	154704	307949	462653	3000000
FundRaising Contract	-	-	-	-	127020	250537	377557	3000000

and therefore the total costs for the user contract, request contract and company contract remain the same before and after privacy integration. However, the costs witnessed for the file contract, student contract, school contract and shareFile contract are more with privacy integration than without privacy. Evidently, this is because the features of privacy incorporation have affected these four contracts. The additional cost of file contract is because it stores an extra hash value on the blockchain which is the original hash value computed by the school over a credential at the time of issuing. Since student and school contracts invoke an instance of file contract, so their costs are also increased. Finally, the increased cost of shareFile contract is attributed to the temporary upload being done by a student to grant private access to the intended company. It is also interesting to note that there is no cost mentioned for CertificateAuthority contract and FundRaising contract before privacy inclusion. This is simply because these contracts are not required at all when privacy is not assimilated.

Table III also depicts that every smart contract has a *gas limit* as well as a *gas price*. Gas limit is the maximum gas that can be spent on a particular contract whereas the gas price is the value of gas represented in the currency of ether. The more the value of gas, the higher is the priority given by the miners and thus the faster will be the execution. Since the sharing of students' credentials is not a delay sensitive scenario, so the gas price used is 1 GWEI for all the experiments carried out. This reduces the operational cost of the overall developed applications thereby making it economically viable.

[10]https://remix.ethereum.org/

TABLE IV: Cost of some of the important functions

Function	Before Privacy					After Privacy				
	Transaction Cost	**Execution Cost**	**Total Cost** (Gwei)	**Total Cost** (Ether*)	**Total Cost** (USD†)	**Transaction Cost**	**Execution Cost**	**Total Cost** (Gwei)	**Total Cost** (Ether*)	**Total Cost** (USD†)
Enrolling a student	22680	49560	72240	0.00007224	0.037	22680	49560	72240	0.00007224	0.037
Uploading a credential#	29912	128601	158513	0.000158513	0.081	33560	189574	223134	0.000223134	0.114
Company retrieving a credential#	24472	134027	158499	0.000158499	0.081	24472	134027	158499	0.000158499	0.081
Student sharing a credential#	23256	1575	24831	0.000024831	0.013	35032	189699	224731	0.000224731	0.115
Set Public key on blockchain	-	-	-	-	-	26264	2154	28418	0.000028418	0.015

depends on title and description length (20 letters are used for both)

* 1 ether = 10^9 gwei, † 1 ether=$ 510.43 on 20.11.2020

Table IV shows the costs (calculated by remix) for some of the important functions without privacy and with privacy incorporation. The difference between the costs shown in Table III and Table IV is that, the former represents the gas consumed to deploy the smart contracts on the blockchain and is merely a one-time cost. Whereas, the later represents the transaction cost for running some of the important functions called after the smart contracts are deployed.

As apparent from Table IV costs for enrolling a student and company retrieving a credential is the same before and after security inclusion. However, costs for uploading a credential is higher with privacy because for realizing this function, the file contract is called which has an extra (original) hash value. The cost of sharing a credential is very high after privacy as compared to the cost before privacy. This is due to the fact that in order to have a private sharing of credentials with an intended company, a student makes a temporary upload of those credentials' metadata after encrypting them with the public key of the company. More specifically, without privacy, granting access of credentials to an intended company requires the student to share merely the location (i.e. an index of integer type) of metadata of the credential on the blockchain with the intended company. In contrast to that, with privacy inclusion, the student has to re-upload the metadata of the newly encrypted credential which contains two hashes of 256 bits as well as title and description of 20 characters each. Finally, there is no cost mentioned for the function setting of public key on the blockchain since the use of the pair of public and private keys

is not there when privacy is not incorporated.

B. Execution Time for User Requests

The total execution time is made up of communication time, block mining time and execution time of the relevant smart contract(s). Since the DApp is built for an academic information system it may have to cater larger amounts of requests, so it is interesting to explore the overall execution time for user requests. A user can send two types of requests to the blockchain via DApp. They are reading data from the blockchain and writing data to the blockchain. Reading data does not involve a creation of any block hence it takes negligible time. However, when a user writes data on the blockchain, transactions are validated and blocks are mined hence it takes some amount of time. The average time for a transaction/request to get processed on the Rinkeby Test Network is 15 seconds according to their official website[11]. To evaluate the performance of the developed DApp, two different types of tests are performed.

1) Execution Time to Upload a Credential: In a privacy protected version of DApp, uploading a credential requires a school to perform the following tasks: (1) compute the original hash of the credential, (2) get the public key of a student from blockchain and encrypt the credential with that key, (3) upload the encrypted credential on the IPFS and get the IPFS hash, and (4) upload the metadata on the blockchain. However, if privacy is not included, then tasks 1 and 2 are not performed. From the blockchain performance point-of-view there will be no change in the execution time to upload a credential either with privacy or without privacy. The reason being Task 1 is performed locally (i.e. at the front-end of DApp) whereas Task 2 is simply a reading operation from the blockchain. Thus whatever is the change observed in the execution time for uploading a credential without privacy and with privacy, this is due to the variation in the validation time taken by the blockchain. To find out the average time and the variations in the execution time, 100 iterations were performed by sending back-to-back requests for uploading a credential. Figure 14 shows the results obtained using the 95% confidence interval. The average turns out to be 16.00337 s whereas the official website claims it to be 15 s. The difference between the observed average time and the official average time is 1.00337 s. This additional delay encountered is due to the execution of smart contracts and time taken by Ethereum to validate and mine new blocks as well as the communication delay.

2) Execution Time to Send Access Request and Grant Access Right: Figure 15 shows the execution time for sending an access request and granting access right, before and after privacy inclusion. In a privacy protected DApp, sending an access request and granting the right involves the following tasks:

[11]https://www.rinkeby.io/#stats

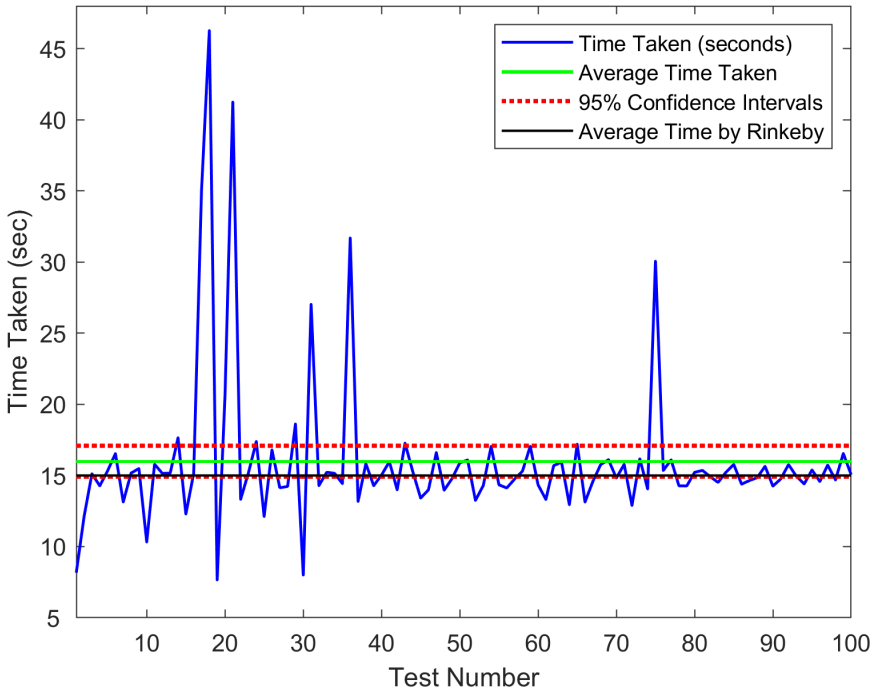
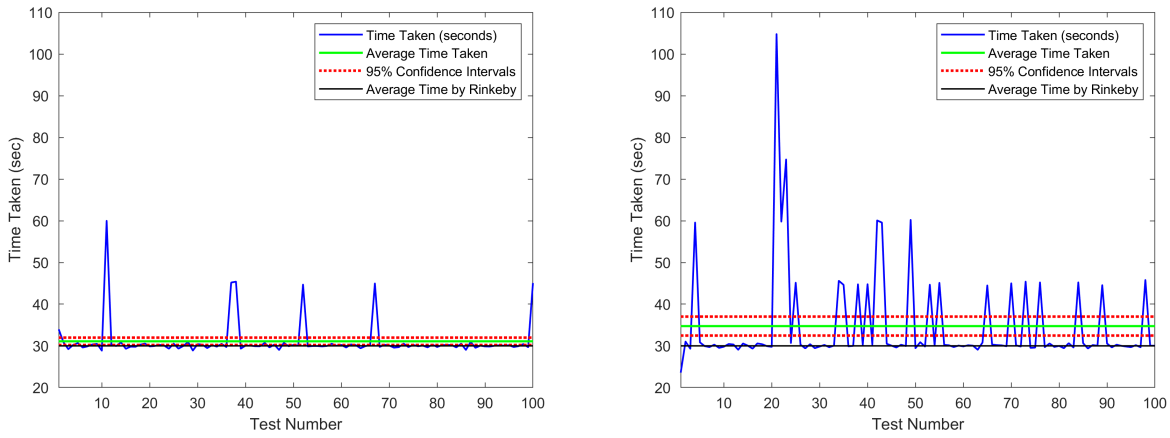


Fig. 14: Execution time for credential upload requests by the school

(1) the company sends an access request to a student, (2) the student views the request and approves it which is followed by encrypting the requested credential with the public key of the company, (3) the student then uploads the encrypted credential on the IPFS and get the IPFS hash, and (4) the student makes a temporary upload of metadata on the blockchain. However, if privacy is not included then tasks 2 and 3 are not performed. Moreover, Task 2 is performed locally at the client side so it takes negligible amount of time. Furthermore, Task 3 (which is uploading a file on IPFS) also takes marginal amount of time as compared to the time required by Ethereum for one write operation. As presented in [34], the average upload time and the average download time in IPFS for the file size equal to 1 MB, are both being less than 0.2 s. Thus, the execution time to send an access request and grant the access right with and without privacy would be almost the same. Any change whatsoever being witnessed depends on the time the blockchain takes to perform two write operations, which corresponds to Task 1 and Task 4. To perform the test, first a company sends a request to a students account. Second, the student views the request and then grants the access. Both requests were pipe-lined to calculate the total time taken. Here, the assumption is that the student immediately approves all the incoming requests. Further, to find out the average time and the variations in the execution time, 100 iterations of such requests were pipe-lined.



(a) Before Privacy

(b) After Privacy

Fig. 15: Execution Time to Send Access Request and Get Access Right

The average time without privacy turns out to be 31.09165 s whereas with privacy it is 34.74195 s. Since the official website claims the time to process each request is 15 s, so for two write operations it should be 30 s. Hence, the difference between the observed average time and the official average time with privacy is 4.74195 s and without privacy is 1.09165 s. Although observations imply that the difference is higher with privacy than without privacy, this is merely because of varying performance of Ethereum and communication delay. It is worth to note that for a single access request by a company demanding access for x number of credentials, it will result in a total $x + 1$ number of write operations to be performed on the blockchain. That is 1 write operation for sending one access request and x number of write operations for uploading metadata of x number of requested credentials.

C. Scalability of the System

For an academic information system that deals with issuing, viewing and sharing of students' credentials with large number of users, scalability is an important requirement. Thus, to explore how the DApp scales with the increasing number of requests, two tests are performed. Both tests, uploading of a credential as well as request for sending access and granting access to credential were carried out asynchronously. Here, asynchronous means many requests (i.e. up to 1000) were sent at the same time from the DApp to Ethereum so as to ascertain the normal working of the application and related smart contracts. Rinkeby Test Network can process approximately 20 transactions/second and a block is mined in about 15 s. Thus, all the transactions received (including the ones sent by our DApp) get processed sequentially in batches of the maximum possible transactions that can be accommodated in a single block. Hence, there

is a queuing of requests if they are more than what a block can hold. This leads to an increase in the execution time with the increase in the total number of requests being sent as shown in Figure 16. It is worth noticing that processing times of transactions are somewhat multiples of 15 since block creation time is approximately 15 s. The increase in time can be normalised if a different consensus algorithm other than PoW is used. This can be achieved by using other platforms which have high computational capabilities and support time-efficient consensus algorithms such as hyperledger [35].

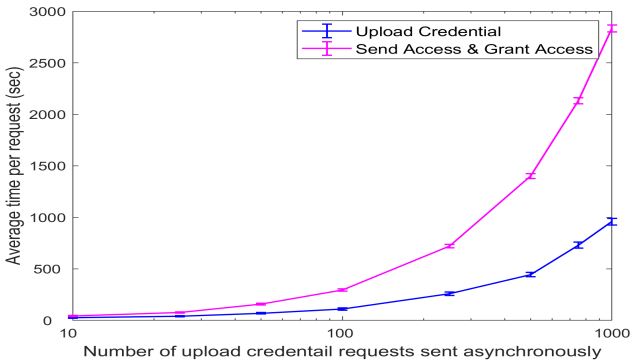


Fig. 16: Scalability test: requests sent asynchronously

VI. Security Analysis of the Implementation

This section aims to analyze the security of the implementation and also presents a discussion on how the developed DApp can withstand the most critical vulnerabilities and attacks.

A. Use of Ethereum to enhance security

Ethereum associates a pair of public and private keys for each user account. The unique Ethereum account address assigned to every user is actually a hashed version of this public key. To use the DApp, users must be connected to the Ethereum network using their private key. Without a valid private key, the account is inaccessible and hence an attacker cannot impersonate any user.

All the transactions performed by a given user are associated with this account address and are stored on the blockchain. When a transaction is made, the transaction is signed by the sender and is sent to the miners for validation. Since the requirements for a transaction (to be made) are described in the smart contract, the transaction gets validated soon after these requirements are met. After validation, the transaction is stored on the Ethereum as an immutable block. For example, the uploadfile method of File contract, which is used by the school to upload a credential has two requirements. First, the logged-in user must be a school to call this method (i.e the sender can only be a school), and second, a school

can upload credential(s) exclusively for those students who are enrolled under it. Thus inherent security features of Ethereum along with carefully designed smart contracts do not leave any scope for an attacker.

B. Various vulnerabilities and their countermeasures

TABLE V: OWASP vulnerabilities [36] and their countermeasures instilled in DApp

Attack	**Description**	**Countermeasure**
Injection	Implies injection of malicious scripts that get executed without authentication.	No use of traditional databases for data storage rules. Moreover, functions/methods can only be called by authenticated users according to the deployed smart contract.
Broken Authentication and Session Management	Weak authentication of session and its management allow the attacker to steal passwords or keys to mount an impersonation attack.	At the time of transaction the logged-in user is verified and thereafter the transaction is processed. Further, since the authentication is done using Metamask which in-turn uses blockchain technology, passwords and keys cannot be compromised unless a user is willingly to do so.
Sensitive Data Exposure	Implies handling and storing of sensitive data in an insecure way thereby making an application vulnerable.	All transactions happening between user and blockchain infrastructure are encrypted as well as the credentials uploaded to IPFS are also encrypted.
XML External Entity	Frail evaluation of referenced external entities makes an application susceptible to malicious intrusion.	Use of solidity language for building smart contracts and use of JSON format for external entity referencing prevents such security risks.
Broken Access Control	Absence of required access control of critical functionalities permits an attacker to access restricted functions and data.	Every function or method of the deployed smart contracts, has a validator that checks if this functionality is accessible to the logged-in authenticated user or not.
Security misconfiguration	Denotes insecure implementation and handling of control plane operations and messages.	Smart contract interactions are well tested using the Remix IDE. Session management by Metamask as well as by the client side application makes the system more secure.
Cross-Site Scripting	Missing input validation & Poor coding practices enable an attacker to insert client-side malicious scripts.	State-of-art coding practices are followed with the use of solidity and Next js. Input validations are performed before sending data to the blockchain, thereby eliminating chances of cross-site scripting.
Insecure deserialization	Indicates remote execution of malevolent code by an attacker to expose privileges of serialized objects and/or corrupt them.	First the remote execution of such attack gets logged in on the blockchain and second if the logged-in user is not authorized then s/he will not have accessibility to functions of the developed smart contracts.
Using Components with Known Vulnerabilities	Vulnerabilities of used open source or third party software, open latent doors for attackers.	Widely used open source trustable software/platform such as Web3js, Next js and Ethereum are used to develop the DApp.
Insufficient Logging and Monitoring	Insufficient logging & Ineffective monitoring leads to delayed (or no) detection of security breaches.	Since blockchain records each and every transaction intensively, logging and monitoring is thus an intrinsic salient feature.

Some of the most popular vulnerabilities in the realm of web applications are identified by the Open Web Application Security Project (OWASP) [36]. Most of these vulnerabilities (nine out of the top

ten) are pertinent to blockchain ecosystem as well [37]. Thus, the intent of this section is to illustrate the appropriate countermeasures which are instilled in the developed DApp to withstand or combat against OWASP vulnerabilities. Table V summarises the top ten attacks, their brief description and countermeasures.

1) Injection: Such attacks happen when malicious scripts or commands are injected by an attacker, which get executed without authentication (for example SQL injection, SSI injection, OS command injection, etc. [38]). In general, parameterized queries and security testing are used to mitigate such attacks. In view of this attack, the developed DApp makes no use of traditional databases for data storage which rules out the possibility of this security risk. Moreover, functions/methods can only be called by authenticated users according to the deployed smart contract.

2) Broken Authentication and Session Management: Insufficient authentication checks and weak session management can give room to intruders to steal login credentials. In general MFA (Multi-factor authentication) methods are used to overcome such attacks. For the developed DApp, at the time of any transaction the logged-in user is always verified and then the transaction is sent. Moreover, the use of Metamask and Ethereum blockchain technology does not allow such attacks unless a user is willingly to do so.

3) Sensitive Data Exposure: Insecure handling and storing of sensitive and private data render an application vulnerable to attacks. In general encryption techniques are very handy to tackle such attacks. For the developed DApp, all the transactions happening between user and blockchain infrastructure are encrypted thereby ensuring data protection. Moreover, the credentials being uploaded on the IPFS are also encrypted using the public key of the intended user. This makes the entire system privacy protected.

4) XML External Entity: Invoking the referenced external malicious entities can lead to serious issues. In general, proper security testing is to be carried out to reveal the possibilities of such attacks, for example SAST [39]. For the developed DApp, the use of solidity language for building smart contracts as well as the use of JSON format for external entity referencing prevents attacks.

5) Broken Access Control: Absence of proper access-restriction checks allows intruders to impersonate an authentic user and access the forbidden contents as well as change them. In general, in-depth testing of an application needs to be carried out to discover and purge such unauthorized accesses. For the developed DApp, every function (or method) of the deployed smart contracts, has a validator that checks if the logged-in user has accessibility to the called functionality.

6) Security Misconfiguration: Such security threat arises when control plane operations and messages are not carefully handled and also when the required security enhancement is not performed as required. In general, thorough security testing of the web application is carried out to find such misconfigurations,

for example DAST [39]. For the DApp, all the interactions among the smart contracts are well tested against possible requests before integration using the Remix IDE. Moreover, sessions are maintained by Metamask as well as the client side application which makes the session management more secure.

7) Cross-Site Scripting: One of the most popular attacks is cross-site scripting. Using such attacks malicious client-side scripts can be injected which can hijack the redirection mechanism and land the users to insecure places. In general, such attacks are tackled using strong input validations and encrypting data. The developed DApp ensures proper input validations before uploading data on the blockchain and state-of-art coding practices are followed.

8) Insecure Deserialization: Leveraging such kind of attacks, an intruder can remotely corrupt or modify serialized objects. In general, security testing tools can help uncover such attacks. In context to developed DApp, if an attacker tries to execute a malicious code remotely then first the attempt will be logged on the blockchain and second since the user is not authorized, s/he will not have accessibility to functions of the developed smart contracts.

9) Using Components With Known Vulnerabilities: Causal use of open source and third-party software enables attackers to use such software's loopholes to get into a system. In general, in-depth static analysis [40] can help understand the vulnerabilities of the modules used. For the developed DApp, widely used open source trustable software/platforms such as Web3js, Next js and Ethereum are used.

10) Insufficient Logging and Monitoring: Knowing the level of logging and monitoring installed in an application, an attacker can easily plan its moves. In general, a comprehensive analysis of an application is carried out and accordingly logs are maintained wherever necessary. For the developed DApp, since blockchain already records each and every transaction intensively thus logging and monitoring is not to be worried.

Since the smart contract can also be used fraudulently and can undermine the performance efficiency [41], there is a need to ensure the correctness of the developed smart contracts. In this regard, some of the possible attacks specific to smart contracts and their corresponding prevention are shown in Table VI.

VII. Conclusions

To resolve the cumbersome task of secured sharing of students' credentials, blockchain technology is leveraged. In this direction, a simple and viable architecture is proposed which enumerates various stakeholders, their roles and the core functionalities of the architecture. Further, a novel mechanism to integrate privacy in the proposed architecture is presented and analyzed as well. For the proof-of-concept of the proposed privacy protected architecture, a prototype, i.e. a decentralized application (DApp) is

TABLE VI: Smart contract related attacks and vulnerabilities [42], [43], [44], with the countermeasure

Attack	**Description**	**Countermeasure**
The DAO attack	The attacker uses a separate smart contract (decentralized autonomous organisation) to call vulnerable fall back functions in the target smart contract using its deployed address.	The developed DApp does not store ether on smart contracts and does not contain any payment functions in the contracts. Further, the use of gas limits in the front-end of DApp prevent fallback functions from getting executed.
Overflow and Underflow	If there is an integer underflow in a smart contract, it results in an integer of 2^{256-1} and if there is an overflow it resets to 0. Hence, this can be used as a vulnerable point of attack.	The developed DApp does not make use of any function which has parameters leading to overflow or underflow.
Parity Multi-Sig Wallet Attack	Multi-sig wallet is a public library, which uses smart contracts to create wallets. The attacker finds a way to initialise the library and gain ownership rights.	Such functions that allow re-initialization are not being used in the current DApp. Contracts once deployed cannot be reinitialised.
Rubixi Attack	The attack occurred on the Rubixi contract provides an investment service where the developers changed the contract name but forgot to change the constructor name, hence the constructor became a public function which could be called by anyone to gain ownership of the contract.	All the contracts' code have been cross validated and appropriate access modifiers have been assigned to all the functions.
Short Address Attack	Ethereum Virtual Machine includes zeroes at the end of an underflow address to ensure the 256 bit data type. However this can be vulnerable if an attacker omits the bits knowingly.	This issue has been fixed by Ethereum and is no longer a vulnerability for versions above 0.5.0. The developed DApp uses version 0.5.0.

developed. Nine smart contracts are designed and deployed as the key constituents of the DApp. Testing and validation of the DApp are carried out using the Ethereum blockchain. Moreover, experiments carried out to compute the costs show that the DApp is economically viable. Also, the experiments conducted to evaluate the execution time of important functions demonstrate that the performance of DApp remains more or less the same without or with privacy integration. Thus the DApp makes all the students' credentials tamper-proof, immutable, authentic, non-repudiable and easy to share. In future, we intend to deploy the architecture over a permissioned blockchain platform to compare its performance. Also, we look forward to integrating the step of revoking credentials, which is at times required.

REFERENCES

[1] A. Singh and K. Chatterjee, "Cloud security issues and challenges: A survey," *Journal of Network and Computer Applications*, vol. 79, pp. 88–115, 2017.

[2] D. Berdik, S. Otoum, N. Schmidt, D. Porter, and Y. Jararweh, "A survey on blockchain for information systems management and security," *Information Processing & Management*, vol. 58, no. 1, p. 102397, 2021.

[3] S. F. Wamba and M. M. Queiroz, "Blockchain in the operations and supply chain management: Benefits, challenges and future research opportunities," *International Journal of Information Management*, vol. 52, p. 102064, 2020.

[4] C. Esposito, M. Ficco, and B. B. Gupta, "Blockchain-based authentication and authorization for smart city applications," *Information Processing & Management*, vol. 58, no. 2, p. 102468.

[5] T. Hardin and D. Kotz, "Amanuensis: Information provenance for health-data systems," *Information Processing & Management*, vol. 58, no. 2, p. 102460, 2020.

[6] C. Oham, R. A. Michelin, R. Jurdak, S. S. Kanhere, and S. Jha, "B-ferl: Blockchain based framework for securing smart vehicles," *Information Processing & Management*, vol. 58, no. 1, p. 102426, 2020.

[7] H. Baniata, A. Anaqreh, and A. Kertesz, "Pf-bts: A privacy-aware fog-enhanced blockchain-assisted task scheduling," *Information Processing & Management*, vol. 58, no. 1, p. 102393.

[8] J. Li, J. Wu, G. Jiang, and T. Srikanthan, "Blockchain-based public auditing for big data in cloud storage," *Information Processing & Management*, vol. 57, no. 6, p. 102382, 2020.

[9] E. Mengelkamp, B. Notheisen, C. Beer, D. Dauer, and C. Weinhardt, "A blockchain-based smart grid: towards sustainable local energy markets," *Computer Science-Research and Development*, vol. 33, no. 1-2, pp. 207–214, 2018.

[10] Q. Chen, G. Srivastava, R. M. Parizi, M. Aloqaily, and I. Al Ridhawi, "An incentive-aware blockchain-based solution for internet of fake media things," *Information Processing & Management*, vol. 57, no. 6, p. 102370, 2020.

[11] B. Putz, M. Dietz, P. Empl, and G. Pernul, "Ethertwin: Blockchain-based secure digital twin information management," *Information Processing & Management*, vol. 58, no. 1, p. 102425.

[12] G. Albeanu, "Blockchain technology and education," *On Virtual Learning*, p. 271, 2017.

[13] A. Grech and A. F. Camilleri, "Blockchain in education," 2017.

[14] R. A. Mishra, A. Kalla, N. A. Singh, and M. Liyanage, "Implementation and analysis of blockchain based dapp for secure sharing of students' credentials," in *2020 IEEE 17th Annual Consumer Communications & Networking Conference (CCNC)*. IEEE, 2020, pp. 1–2.

[15] L. J. Børresen, E. Meier, and S. A. Skjerven, "Detecting fake university degrees in a digital world," in *Corruption in Higher Education*. Brill Sense, 2020, pp. 102–107.

[16] P. Voigt and A. Von dem Bussche, "The EU General Data Protection Regulation (GDPF)," *A Practical Guide, 1st Ed., Cham: Springer International Publishing*, 2017.

[17] T. Hewa, M. Ylianttila, and M. Liyanage, "Survey on blockchain based smart contracts: Applications, opportunities and challenges," *Journal of Network and Computer Applications*, p. 102857, 2020.

[18] G. Chen, B. Xu, M. Lu, and N.-S. Chen, "Exploring blockchain technology and its potential applications for education," *Smart Learning Environments*, vol. 5, no. 1, p. 1, 2018.

[19] M. Sharples and J. Domingue, "The blockchain and kudos: A distributed system for educational record, reputation and reward," in *European Conference on Technology Enhanced Learning*. Springer, 2016, pp. 490–496.

[20] R. Arenas and P. Fernandez, "Credenceledger: a permissioned blockchain for verifiable academic credentials," in *2018 IEEE International Conference on Engineering, Technology and Innovation (ICE/ITMC)*. IEEE, 2018, pp. 1–6.

[21] O. Andreev and H. Daskalov, "A framework for managing student data through blockchain," in *Proceedings of X International Scientific Conference E-governance and e-Communications*, 2018.

[22] A. Young and S. Verhulst, "Creating Immutable, Stackable Credentials Through Blockchain at MIT," 2018.

[23] M. Han, Z. Li, J. He, D. Wu, Y. Xie, and A. Baba, "A novel blockchain-based education records verification solution," in *Proceedings of the 19th Annual SIG Conference on Information Technology Education*, 2018, pp. 178–183.

[24] A. Srivastava, P. Bhattacharya, A. Singh, A. Mathur, O. Prakash, and R. Pradhan, "A distributed credit transfer educational

framework based on blockchain," in *2018 Second International Conference on Advances in Computing, Control and Communication Technology (IAC3T)*. IEEE, 2018, pp. 54–59.

[25] E. E. Bessa and J. S. Martins, "A blockchain-based educational record repository," *arXiv preprint arXiv:1904.00315*, 2019.

[26] J. Hope, "Give students ownership of credentials with blockchain technology," *The Successful Registrar*, vol. 19, no. 1, pp. 1–7, 2019.

[27] T. Kanan, A. T. Obaidat, and M. Al-Lahham, "Smartcert blockchain imperative for educational certificates," in *2019 IEEE Jordan International Joint Conference on Electrical Engineering and Information Technology (JEEIT)*. IEEE, 2019, pp. 629–633.

[28] T. Arndt and A. Guercio, "Blockchain-Based Transcripts for Mobile Higher-Education," *International Journal of Information and Education Technology*, vol. 10, no. 2, 2020.

[29] T. Nguyen, "Gradubique: An academic transcript database using blockchain architecture," 2018.

[30] M. Turkanović, M. Hölbl, K. Košič, M. Heričko, and A. Kamišalić, "EduCTX: A blockchain-based higher education credit platform," *IEEE Access*, vol. 6, pp. 5112–5127, 2018.

[31] W. Gräther, S. Kolvenbach, R. Ruland, J. Schütte, C. Torres, and F. Wendland, "Blockchain for education: lifelong learning passport," in *Proceedings of 1st ERCIM Blockchain Workshop 2018*. European Society for Socially Embedded Technologies (EUSSET), 2018.

[32] P. Ocheja, B. Flanagan, H. Ueda, and H. Ogata, "Managing lifelong learning records through blockchain," *Research and Practice in Technology Enhanced Learning*, vol. 14, no. 1, p. 4, 2019.

[33] Q. Zhao, S. Chen, Z. Liu, T. Baker, and Y. Zhang, "Blockchain-based privacy-preserving remote data integrity checking scheme for IoT information systems," *Information Processing & Management*, vol. 57, no. 6, p. 102355, 2020.

[34] S. Muralidharan and H. Ko, "An InterPlanetary File System (IPFS) based IoT framework," in *2019 IEEE International Conference on Consumer Electronics (ICCE)*. IEEE, 2019, pp. 1–2.

[35] X. Xu, G. Sun, L. Luo, H. Cao, H. Yu, and A. V. Vasilakos, "Latency performance modeling and analysis for hyperledger fabric blockchain network," *Information Processing & Management*, vol. 58, no. 1, p. 102436.

[36] "OWASP TOP 10 VULNERABILITIES," The Open Web Application Security Project (OWASP), 2017, https://www.veracode.com/directory/owasp-top-10 accessed on 2019-06-20.

[37] H. Poston, "Mapping the OWASP Top Ten to Blockchain," *Procedia Computer Science*, vol. 177, pp. 613–617, 2020.

[38] L. K. Seng, N. Ithnin, and S. Z. M. Said, "The approaches to quantify web application security scanners quality: a review," *International Journal of Advanced Computer Research*, vol. 8, no. 38, pp. 285–312, 2018.

[39] P. A. I. Team, "How to reveal application vulnerabilities? SAST DAST IAST and others," 2015.

[40] Y.-W. Huang, F. Yu, C. Hang, C.-H. Tsai, D.-T. Lee, and S.-Y. Kuo, "Securing web application code by static analysis and runtime protection," in *Proceedings of the 13th international conference on World Wide Web*. ACM, 2004, pp. 40–52.

[41] T. Hu, X. Liu, T. Chen, X. Zhang, X. Huang, W. Niu, J. Lu, K. Zhou, and Y. Liu, "Transaction-based classification and detection approach for ethereum smart contract," *Information Processing & Management*, vol. 58, no. 2, p. 102462.

[42] N. Atzei, M. Bartoletti, and T. Cimoli, "A survey of attacks on ethereum smart contracts," *IACR Cryptol. ePrint Arch.*, vol. 2016, p. 1007, 2016.

[43] S. Sayeed, H. Marco-Gisbert, and T. Caira, "Smart contract: Attacks and protections," *IEEE Access*, vol. 8, pp. 24 416–24 427, 2020.

[44] P. Praitheeshan, L. Pan, J. Yu, J. Liu, and R. Doss, "Security analysis methods on ethereum smart contract vulnerabilities: a survey," *arXiv preprint arXiv:1908.08605*, 2019.

APPENDIX A
PRELIMINARIES OF ECC ON RELATED PRIMITIVES

The section discusses the preliminaries of ECC, which develops the necessary background to understand the usage of ECC with regard to the 8 different core functionalities mentioned in Section III-A.

ECC allows lightweight public key cryptographic solutions and is based on the algebraic structure of Elliptic Curves (ECs) over finite fields. A curve in the finite field F_p is denoted by $E_{p(a,b)}$ and is defined by the equation $y^2 = x^3 + ax + b$ with a and b two constants in F_p and $\Delta = 4a^3 + 27b^2 \neq 0$. The base point generator of $E_{p(a,b)}$ of prime order q is denoted by G. The two main operations in ECC are addition and multiplication. The addition of two points, $P1 + P2 = R$, results in a new EC point R. The scalar EC multiplication with $r \in F_q$ for a given EC point P is represented by $R = rP = (R_x, R_y)$, with $R_x, R_y \in F_p$, resulting in the point R of the EC. The security of ECC relies on the following two computationally hard problems:

- *Elliptic Curve Discrete Logarithm Problem (ECDLP):* This problem states that given two EC points R and Q of $E_{p(a,b)}$, it is computationally hard for any polynomial-time bounded algorithm to determine a parameter $x \in F_q^*$, such that $Q = xR$.
- *Elliptic Curve Diffie Hellman Problem (ECDHP):* Given two EC points $R = xP, Q = yP$ with two unknown parameters $x, y \in F_q^*$, it is computationally hard for any polynomial-time bounded algorithm to determine the EC point xyP.

1) Encryption and decryption: Thanks to the ECDHP, the construction of a common secret key between sender and receiver is very simple. Here, (d_S, Q_S) and (d_R, Q_R) denote the key pairs of sender and receiver respectively. A common shared session key K at timestamp TS equals to $K = H(d_S Q_R \| TS) = H(d_R Q_S \| TS)$, which is only derivable by sender and receiver respectively. We denote the encryption of message M using key K in order to obtain the ciphertext C by $C = E_K(M)$.

In case the sender just want to perform a public key encryption for receiver R, without actual authentication of itself, the Elliptic Curve Integrated Encryption Scheme (ECIES) algorithm is the most efficient. To this end, the sender chooses a random value r and computes $U = rG$. The encryption key K is now derived as $K = H(rQ_R)$ and the message to be sent to the receiver consists of $C = E_K(M)$ together with the random EC point U.

2) Signature generation and verification: Again, (d_S, Q_S) and (d_R, Q_R) denote the key pairs of sender and receiver, respectively. The Schnorr signature generation of S by ID_S on message M requires the following steps.

- Choose random value r and compute $R = rG$.

- Compute $h = H(M\|R)$ and $s = r - hd_S$

The tuple (M, h, s) is then sent to the receiver, which can now verify the authenticity of it by means of the following steps.

- Compute $R = sG + hQ_S$.
- Check if $H(M\|R) == h$.

APPENDIX B
ELLIPTIC CURVE QU VANSTONE (ECQV) CERTIFICATES FOR SIGN-UP PROCESS

In order to use ECC in a public key based system for encryption and signature generation, the sender and receiver possess a private/public key pair defined by (d, Q) where $Q = dG$, with G the system parameter representing the generator point of the curve. Certificates are required in order to link the public key to the identity of its owner. The ECQV certificate mechanism defines a very lightweight solution and could be used during the sign-up process. ECQV makes it possible for a user to generate a key pair based on a received certificate which is constructed by a trusted specialized government body. Using this mechanism, only the user who signs-up knows the private key whereas any other entity, having the unique ID and the certificate of a user, is able to generate only the public key of that user.

The secret key pair of the trusted government body is denoted by (d_{govt}, Q_{govt}). The different steps in the derivation of the key pair (d_u, Q_u) for a user with identity ID_u are described in Figure 17. Here H represents a hash function, like e.g. SHA2 or $SHAKE128(M, 256)$, having a 256-bit output length and 128-bit overall security.

Note that any other user can derive the public key of a user with identity ID_u given its certificate $cert_u$ by

$$Q_u = H(cert_u\|ID_u)cert_u + Q_{govt}.$$

This follows from the fact that

$$\begin{aligned} Q_u &= d_uG = (H(cert_u\|ID_u)r_u + r)G \\ &= H(cert_u\|ID_u)r_uG + rG \\ &= H(cert_u\|ID_u)R_u + (H(cert_u\|ID_u)r_T + d_{govt})G \\ &= H(cert_u\|ID_u)R_u + H(cert_u\|ID_u)R_T + Q_{govt} \\ &= H(cert_u\|ID_u)cert_u + Q_{govt} \end{aligned}$$

Entity requesting key material		Government Body
Choose $r_u \in_R \mathbb{F}_p, R_u = r_u G$		
	$\xrightarrow{ID_n \| R_n}$	
		Choose $r_T \in_R \mathbb{F}_p, R_T = r_T G$
		$cert_u = R_u + R_T$
		$r = H(cert_u \| ID_u) r_T + d_{govt}$
	$\xleftarrow{r \| cert_u}$	
$d_u = H(cert_u \| ID_u) r_u + r$		
$Q_u = d_u G$		

Fig. 17: Steps and computations of the ECQV implicit certificate based key construction