

Degree Validation Application Using Solidity and Ethereum Blockchain

Chafic BouSaba
CTIS Department
Guilford College
Greensboro, NC 27410
bousabacw@guilford.edu

Ethan Anderson
CTIS Department
Guilford College
Greensboro, NC 27410
andersones@guilford.edu

Abstract— One of the most common actions for students post-graduation is requesting a **degree validation** or an official transcript of grades. Requests for degree validations happen for various reasons such as an **employment requirement**, a pursuit of a **higher degree education**, **foreign degree accreditation**, or even for a personal reason. Those who are requesting the degree validation are usually on a time constraint. **Traditional methods** of degree validation are often complicated, **time consuming**, **require manpower** to maintain, and potentially have security or privacy issues. We present a software application implementing an automatic degree validation system that uses the Ethereum, an open-source, public, blockchain-based **distributed computing** platform, and that features smart contract functionality. The application is developed using Solidity, a contract-oriented programming language, and is run by the Ethereum Virtual Machine (EVM). This application allows students and universities to automatically verify each other's identity while maintaining transaction validity and referential integrity through a blockchain ledger and maintaining security through PGP key authentication. The university checks the student's request against its own secure MongoDB database. Our application also generates the student's Ethereum wallet using the Truffle development Ethereum network. Truffle generates the student's private and public keys. Our Application pulls the student public key, hashes the last 20 bytes, generates Keccak256 Hash, and stores it on the MongoDB as the student Ethereum Address. To ensure the information's privacy, the blocks of the blockchain are mainly formed of two encrypted fields using the Elliptic Curve Integrated Encryption Scheme (ECIES) library. The first field is encrypted using the student's public key and the second field is encrypted using the third party public field. Third parties, such as employers or other higher education institutions, **are able to check its encrypted field of the block using its own private key**, once the degree is validated or the transcript is generated. Results of testing show our application to be scalable, secure, have efficient gas costs on the Ethereum network, and minimize human interaction.

Keywords—Ethereum, Solidity, blockchain

I. INTRODUCTION

The astronomical rise of cryptocurrency at the end of 2017 has led many companies, of various sectors, to invest in one of its revolutionary backbone technologies, the blockchain, to find other potential uses for it. A blockchain is a type of distributed ledger in which data is stored in blocks 'chained' together using cryptographic signatures enabling users to corroborate the accuracy of the data and participate in transactions tied to the data [1]. With the amount of development on blockchain and distributed cryptographic applications, organizations are realizing the utility of

cryptographic technologies to handle transactions and protect user's security. One such organization that we identified is that of a university, in which a plethora of students and employees share information and interact daily. Potential issues with universities' information systems include user privacy, the need for third party platforms, and human error. By creating a blockchain platform in the educational space, we create the opportunity for universities to update their existing software systems as well as create the potential for a new blockchain based educational system

A blockchain is a core technology for cryptocurrency. It could be defined as a distributed, decentralized, public ledger. "Blocks" on the blockchain are stored digital information, up to 1 MB of data, stored linearly and chronologically to solve many issues of **security, integrity, and trust**. A "block" consists of three parts: (a) information about transactions, for example the date and time, (b) information about who is participating in transactions, the student's name and major, and (c) information that distinguishes them from other blocks, a unique generated code called a "hash".

We submit a solution to these system using Ethereum smart contracts and a **decentralized application** (DApp). Ethereum offers a blockchain based virtual machine, called the Ethereum Virtual Machine (EVM), which allows for smart contract functionality. Smart contracts, systems which "automatically move digital assets according to pre-specified rules" allow businesses to organize their information and allow other parties to interact with them in an automated and secure way [2]. Smart contracts differ from traditional contracts in that they integrate with existing software systems automatically and do not require human interaction. These contracts allow for transactions to interact with the Ethereum network and for information to be exchanged between different parties.

Our software solution implements a user profile application using Ethereum Smart contract functionality and a web-based graphical interface. The blockchain technology Ethereum connects to the EVM and allows for transactional logic to be programmed into the blockchain using the Solidity programming language. The implementation of our application is summarized in Figure 1. Students, the users of the application, can authenticate via an Ethereum public key and private key in order to edit their information on the blockchain. The students' information is encrypted using the students public key and can also be encrypted using the public key of the university or trusted third party.

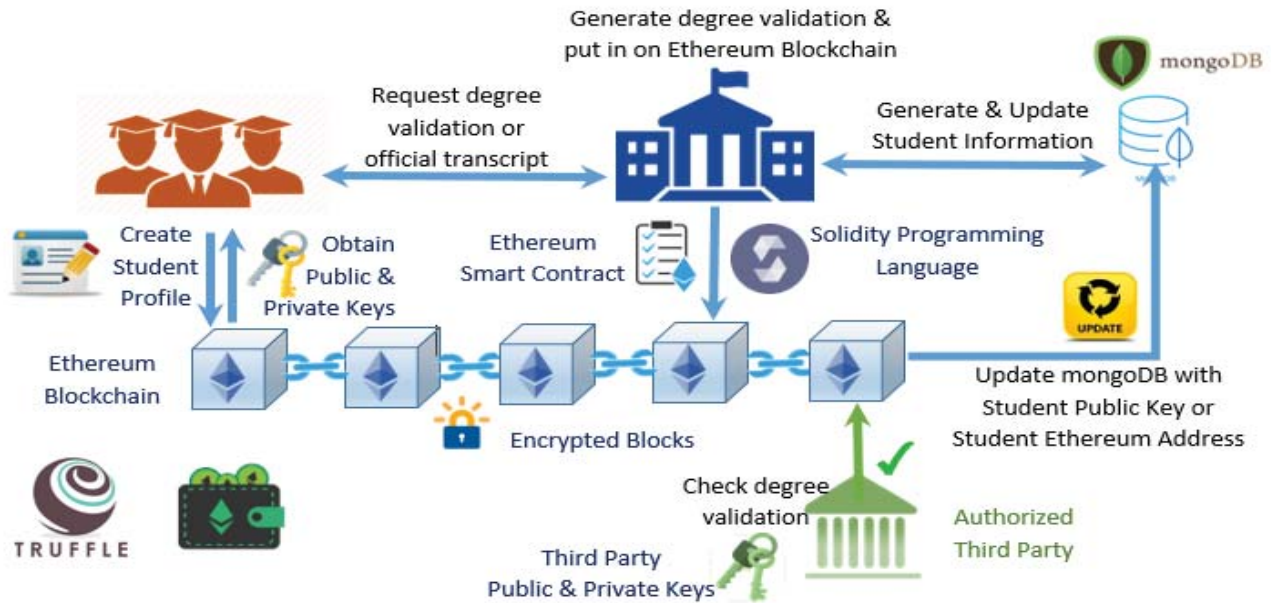


Figure 1 - Application Software Architecture Diagram

Likewise, students, universities and trusted third parties can retrieve student's information by accessing student information and using private key decryption. Private key decryption uses an alphanumeric string given to the user to turn data into an encoded version that can only be decoded using the private key. This allows universities to authenticate student's information automatically, thus constructing the validated blockchain and allowing students and third parties to view the degree. Once authenticated, students can edit their associated profile information. Students information is immutable unless purposefully edited and a history of information edits is stored on the blockchain, thus maintaining integrity and simplifying maintenance.

By allowing for students to authenticate using private and public keys and edit sensitive personal information without the manual input of a university employee, students can quickly and securely get the information they need without involving human error or waiting time while minimizing manpower on behalf of the university. Students, universities and third parties also have pre-defined roles in which they interact with the blockchain, maintaining privacy and lowering the complexity of the degree validation process.

Our software solution also ties into existing information systems by providing interoperability with an open source, secure database. A database is a collection of information organized to allow efficient retrieval [3]. Once students create a new profile and enter in their personal profile information and their Ethereum address, their information is checked against an existing database.

Next, we explore related work to see other implementations of applications that use the Ethereum network to share user information and interact with third parties.

II. RELATED WORK

A team of researchers implemented a sharing DApp, using a Solidity contract, which interacted with a web application. Their implementation allowed users (device owners) to interact with a third party (device renters) and tracked transaction attributes such as deposit, rental price and device description [4]. Their implementation showcases the use of blockchain in the financial sector with multiple types of users. Another team of researchers analyzed the use of blockchain technology in the pharma sector [5]. One of their use-cases examined a shipping platform created by a start-up called Modium.io AG [6], which implemented Solidity contracts that interacted with a PostgreSQL [7] database, an Android client and a variety of sensors. Their implementation showcases the use of blockchain to track the status of inventory in a time-sensitive manner while tracking information related to the supply chain.

The university system shares many of the same problems that businesses and financial institutions do, with the large number of students and third parties that universities must share information with while maintaining data integrity and confidentiality. Furthermore, any solution to data storage would need to build upon existing data storage methods, such as databases and directories.

Our application shared elements with these examples, such as the implementation of a database system and a web-based GUI. We extended upon these examples by implementing profile information encryption and decryption, and interactivity with MongoDB [8]. Our application also demonstrates the potential for blockchain to be used in the education sector to satisfy universities information needs.

Next, we will describe our software application and explain how we implemented our smart contract using Solidity, our MongoDB database, and our web-based user interface.

III. SOFTWARE ARCHITECTURE

The back-end of our application is an Ethereum smart contract, programmed in Solidity on the EVM. The functions of our smart contract interact with a front-end web application in which users can view their profile and edit their personal information. Third parties can also view student's profiles but cannot edit their information. Next, we will go into the frameworks and languages that we used and how we implemented each part of our application.

A. Smart Contract Implementation

Our contract implements functions for verification and profile interaction including *isUser*, *newUser*, *updateUser* and a series of getter functions. Encrypted student profile information is stored in a struct named *StudentInfo* which contains profile attributes (**major, name, graduation year**) and a unique index for each user. Each student is identified by their address, which is then stored in a mapping of Ethereum addresses to *StudentInfo* structs called *students*. **Student's Ethereum addresses are ties to students Ethereum wallets and are associated with their public key.** All the addresses are also stored in an array called *userIndex*. In order to maintain data integrity, each student's profile information is required to be unique and is assigned an individual index. Allowing students to create their own profile allows them to securely access their information and reduces interaction from other parties.

Separately to the *students* mapping, we also created a *thirdPartyStudents* mapping which also contains student profile information. This mapping contains student profile information which is encrypted using the public key of our trusted third-party user. The function *isUser* returns a boolean and takes an Ethereum address as a parameter. It checks whether the input address is already stored in *userIndex*, which would indicate that there is already a user profile with that address. The function returns true if the users address is in use. This function prevents users from creating duplicate profiles in our system.

The functions *updateUser* and *newUser* return a Boolean and take an address and two strings as inputs. Each function emits an event to the Ethereum network. *updateUser* checks whether *isUser* returns true then updates the attributes of *major* and *gradYear*. *updateUser* emits an event named *PushExistingUser* and returns true if it executes successfully. *newUser* checks whether *isUser* returns false then updates the attributes of *major*, *gradYear* and pushes the student's address onto the *userIndex* array. *newUser* emits an event named *PushNewUser* and returns true if it executes successfully. Only students can run these functions and update their profile information once they have authenticated.

The functions *getUserMajor*, *getName*, *getUserGradYear* and *getMaxIndex* return profile attributes but do not change student's profile information and do not emit events. *getUserMajor* takes an address as an input and returns the student's associated major. *getName* takes an address as an input and returns the student's associated name. *getUserGradYear* takes an address as an input and returns

the student's associated graduation year. *getMaxIndex* returns the number of students registered on the blockchain. Both students and trusted third parties can run these functions and view student information provided they can decrypt the information off-blockchain using their private keys. Figure 2 shows a Unified Modeling Language (UML) diagram of these functions and their parameters.

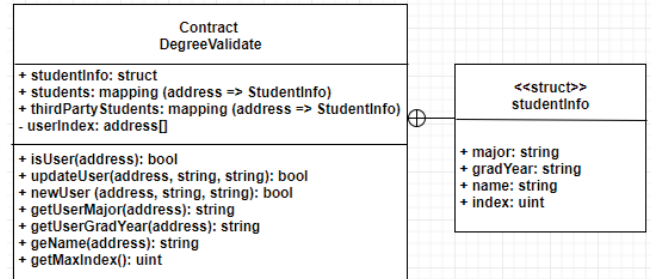


Figure 2 - UML Representation of our Smart Contract Application

The smart contract is deployed using the truffle development Ethereum network, which is included in the truffle framework. The truffle framework is a collection of tools including a testing framework and Javascript bindings that allow us to connect our blockchain assets to external tools such as our web application. Information that is passed to the blockchain is encrypted using elliptic-curve encryption via the *eth-ecies* [8] library, which is a Javascript library that uses elliptic-curve cryptography to encrypt and decrypt data. The information is encrypted using the student's public key as well as a separate public key for third party validation. The info is decrypted off-blockchain using the private keys for the student and third party.

B. Database Implementation

The database for our application is implemented using *MongoDB*. MongoDB is an open-source database which can interact with server side JavaScript, allowing for interactivity with our web application [9]. Student's Ethereum addresses and profile attributes are stored in the database once they have created a wallet and entered their information into our web application. An example document is shown in Figure 3. Our MongoDB database allows for student's profile info to be verified against university records. Whenever a student authenticates to our web application using their Ethereum wallet and enters in profile info, our application inserts the student's Ethereum address alongside their other profile attributes in the database.

```

1 {
2   "_id": {
3     "$oid": "5bd63e57cba18f8653f1a321"
4   },
5   "name": "John Smith",
6   "major": "math",
7   "gradyear": "2016",
8   "address": "0x627306090abab3a6e1400e9345bc60c78a8bef57"
9 }
  
```

Figure 3 - MongoDB Document with Student Profile Information

C. User Interface Implementation

The user interface for our application is implemented in the browser using the *truffle* [10] framework and *web3js* JavaScript library [11]. These libraries allow for us to access our contract using our JavaScript application. Students can enter in their name and choose from a list of majors and graduation years. Solidity functions are called on the client's machine and the information that the user types in is encrypted and passed to the blockchain. To display user's profile information, profile attributes are retrieved from the blockchain and decrypted using student's private key. An image of our user interface is shown in Figure 4. Next, we will summarize the functions in our web application and how they interact with our database and blockchain.

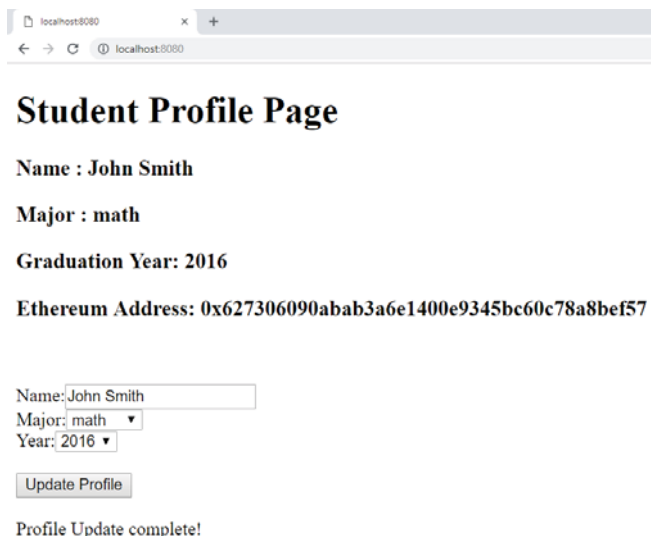


Figure 4 - User Interface for Degree Validation Application

To handle encryption and decryption of the student's private information, we created functions called *encryptMessage* and *decryptMessage*. *EncryptMessage* takes a public key and a string as an input and returns an encrypted string. *EncryptMessage* is used to pass information that the user enters into our web application to the blockchain.

DecryptMessage takes a private key and a string as an input and returns a decrypted string. *DecryptMessage* is used to take information retrieved from the blockchain and decrypt it to be displayed in our browser.

To handle interoperability with our *MongoDB* database, we implemented *insertDB* and *queryDB*. *InsertDB* edits an existing document in our database by adding a student's Ethereum address alongside their corresponding profile information using a HTTP PUT request to our database. *QueryDB* verifies whether a user exists and whether the user's profile information matches the information in the database using a HTTP GET request. Whenever a user first authenticates with their Ethereum wallet to our web application, they are prompted to enter in profile information which is then checked against the database.

To handle interoperability with our blockchain, we implemented functions called *refreshName*, *refreshMajor*,

refreshGradYear and *updateUser*. *RefreshName*, *refreshMajor* and *refreshGradYear* retrieve encrypted profile attributes from the blockchain, decrypts them, then displays them in the browser window. *UpdateUser* takes information that the user inputs into the browser, encrypts it and sends it to the blockchain to be stored. The information that the user sends to the blockchain is checked against a database of student profiles. Next, we will describe the testing that was done to verify that our application was successfully implemented.

IV. RESULTS & DISCUSSION

We created a test contract using *Truffle*'s automated testing framework which tested the practical implementation of our DApp. We tested the creation of a new user profile and updating an existing user profile. One measure of the effectiveness of our tests is gas, which the Ethereum foundation defines as "a measurement roughly equivalent to computational steps" [10] and is automatically measured by the EVM. It costs approximately 110k gas to create a new user and 60k gas to update an existing user. We tested using ten different student accounts on our Ethereum network and the gas cost remained consistent as the number of users increased. We also tested using a third-party account who was able to view, but not edit, student's profile information. Our tests were successful and covered the primary use cases for our application.

When looking at what is next for implementing a Degree Validation application, a next step could include enabling MetaMask [12] to interact with our web application. MetaMask is a browser extension which allows for users to authenticate using their personal Ethereum wallet and sign blockchain transactions. A demonstration of the MetaMask plug-in shown in Figure 5. Implementing MetaMask would allow for universities to let users authenticate on their local devices and run their contracts on a wider variety of Ethereum networks. Another feature to implement would be to store different profile information on the blockchain. While student's name, major and year of graduation are stored, future implementations could store student's class history, grade point average or other pieces of information.

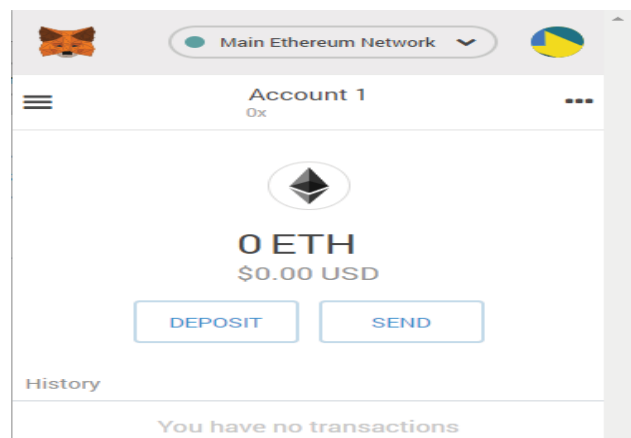


Figure 5 - MetaMask Browser Extension

V. CONCLUSION & FUTURE WORK

In conclusion, our application provides an example for creating a user profile application using the Solidity programming language with a web based front-end. Implementing privacy and security by providing public key encryption and private key decryption as well as minimizing human interaction by implementing smart contract functionality, it provides an example of how to use smart contracts to support the information needs of universities. Future work adds a blockchain for the courses, implementation of students' grades, and tying all that to the registrar's office. Potential other avenues to explore could include payment processing for student's accounts and transfer of academic documents to students.

REFERENCES

- [1] Walport M, "Distributed Ledger Technology: Beyond Blockchain," UK Government Office for Science, Tech. Report, pp. 17/
- [2] V. Buterin, "A Next-Generation Smart Contract and Decentralized Application Platform," *Ethereum Wiki*. [Online]. Available: <https://github.com/ethereum/wiki/wiki/White-Paper>. [Accessed: 20-Oct-2018].
- [3] What is a database? [Online]. Available: https://www.usg.edu/galileo/skills/unit04/primer04_01.phtml. [Accessed: 18-Dec-2018].
- [4] A. Bogner, M. Chanson, and A. Meeuw, "A Decentralised Sharing App running a Smart Contract on the Ethereum Blockchain," *Proceedings of the 6th International Conference on the Internet of Things - IoT16*, pp. 177-178, 2016.
- [5] T. Bocek, B. B. Rodrigues, T. Strasser, and B. Stiller, "Blockchains everywhere - a use-case of blockchains in the pharma supply-chain," *2017 IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*, 2017.
- [6] "modum.io AG," GitHub. [Online]. Available: <https://github.com/modum-io>. [Accessed: 28-Nov-2018].
- [7] "PostgreSQL: The World's Most Advanced Open Source Relational Database," *PostgreSQL: About*. [Online]. Available: <https://www.postgresql.org/>. [Accessed: 28-Nov-2018].
- [8] *ECIES encrypt/decrypt library for Ethereum*, June-2018. [Online]. Available: <https://www.npmjs.com/package/eth-ecies/> [Accessed: 10-Nov-2018].
- [9] *The MongoDB Database*, 06-Nov-2018. [Online]. Available: <https://github.com/mongodb/mongo>. [Accessed: 10-Nov-2018].
- [10] TruffleSuite. 2018. The most popular blockchain development framework. Available: <https://github.com/trufflesuite/truffle>. [Accessed: 20-Oct-2018].
- [11] web3js. 2018. Ethereum JavaScript API. Available: <https://github.com/ethereum/web3.js>. [Accessed: 20-Oct-2018]. "MetaMask Ethereum Browser Extension," MetaMask. [Online]. Available: <https://metamask.io/>. [Accessed: 29-Nov-2018].