**Document Verification using Blockchain: DOC-BLOCK**

By

Yamin Arafat (Roll- 1607032)

Iftekhar Toufique Imam (Roll- 1607040)

A REPORT ON SYSTEM DEVELOPMENT PROJECT

****

**Supervisor:**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Shaikh Akib Shahriyar**

Lecturer, Dept. of CSE, KUET

Khulna University of Engineering & Technology  
Khulna 920300, Bangladesh

December 2019

# Acknowledgement

All praises and greatness go to Allah for His limitless kindness and blessings. Without His desire we would not be here as we are today.

Then, we must sense grateful to and wish to acknowledge our insightful indebtedness to **Mr. Shaikh Akib Shahriyar**, Lecturer of CSE department and our Supervisor of the Project. His endless endurance, scholarly guidance, continual encouragement, constant and lively supervision, constructive criticism, priceless suggestion made it possible to come up to this phase. Without his inspiring, enthusiasm and encouragement, this work could not be completed.

# Abstract

As rapid growth of use of Technology, Digital Document is more convenient than a hard copy of a document. In many actions in life, we are need of verifying these documents. Purpose of our developed system, DOC-Block is to create a fast-reliable secured File share and verification system. In order to verify a document, we need to send it to respectful organization who published it for verification. This is a time-consuming process and is controlled by human, which is not reliable in some case. This is where our solution comes in. The primary purpose of this system is to build up adecentralized web application using Ethereum blockchain technology. This system will allow us to upload various kinds of documents into the blockchain and as blockchain is a peer-2-peer network any uploaded documents will be highly protected which will be difficult to break or manipulate. So, the most important thing about this system is online storage security and with the help of Ethereum blockchain technology we can ensure the safety of personal & important documents on the internet. Next important part of this system is verification of these uploaded documents for any organization or authority. It is easier to find if any document is valid or invalid with just a click. And also, every document has a unique hash value to identify its existence. This hash value is private for only those who are using the current account. By using respective hash value for any document we can download that document in our local storage quickly for checking or any other purpose.

**Table of Contents**

[Acknowledgement 2](#_Toc28132774)

[Abstract 3](#_Toc28132775)

[1. Introduction 6](#_Toc28132776)

[2. Used Software & Dependencies 7](#_Toc28132777)

[2.1 Blockchain 7](#_Toc28132778)

[2.2 Ethereum 8](#_Toc28132779)

[2.3 Solidity 9](#_Toc28132780)

[2.4 Infura 9](#_Toc28132781)

[2.5 IPFS 10](#_Toc28132782)

[3. Methodology to Develop the System 10](#_Toc28132783)

[3.1 Hashing Method SHA-256 10](#_Toc28132784)

[3.2 Security in Decentralized Database 13](#_Toc28132785)

[Why it’s more secured than storing the data in a typical centralized database? 13](#_Toc28132786)

[4. Implementation: 14](#_Toc28132787)

[4.1 Designing the Contract 14](#_Toc28132788)

[Add a document 15](#_Toc28132789)

[Verify a document 15](#_Toc28132790)

[Get document adder info 15](#_Toc28132791)

[Set user name 15](#_Toc28132792)

[4.2 Creating the Web Application 16](#_Toc28132793)

[Check Metamask and web3 16](#_Toc28132794)

[Fetch and set User Info 17](#_Toc28132795)

[Set User Name 17](#_Toc28132796)

[4.3 IPFS for Upload/Download 18](#_Toc28132797)

[5. Application Functionality 18](#_Toc28132798)

[5.1 Verify Document 18](#_Toc28132799)

[Verifying a document 19](#_Toc28132800)

[Generate SHA-256 hash 19](#_Toc28132801)

[5.2 Upload Document 20](#_Toc28132802)

[Adding a Document 20](#_Toc28132803)

[5.3 Download a Verified Document 20](#_Toc28132804)

[6. Result Analysis 20](#_Toc28132805)

[Website Demo 21](#_Toc28132806)

[7. Limitations 25](#_Toc28132807)

[8.Conclusion 25](#_Toc28132808)

[9.Future Plans 25](#_Toc28132809)

[10.References 26](#_Toc28132810)

**List of Figures**

|  |  |  |
| --- | --- | --- |
| **Figure No** | **Caption** | **Page** |
| 1.1.1 | **Document Verify** | **6** |
| 2.1.1 | **Blockchain** | **7** |
| 2.2.1 | **Ethereum** | **9** |
| 2.4.1 | **Infura** | **10** |
| 2.5.1 | **Ipfs** | **10** |
| 3.1.1 | **Jth internal step of the SHA – 256 Compression function C** | **12** |
| 3.1.2 | **SHA -256 message schedule.** | **12** |
| 3.1.3 | **Hash Matching** | **13** |
| 4.1.1 | **Smart Contract** | **14** |
| 5.1.1 | **Verification** | **18** |
| 6.1.1 | **Screenshots** | **21** |
| 6.1.2 | **Adding a document** | **22** |
| 6.1.3 | **Download a document** | **22** |
| 6.1.4 | **Uploading a file for verification** | **22** |
| 6.1.5 | **Verification Info Success** | **23** |
| 6.1.6 | **Verification Info Error** | **23** |

# 1. Introduction

As rapid growth of use of Technology, Digital Document is more convenient than a hard copy of a document. In many actions in life, we are need of verifying these documents. In order to verify a document, we need to send it to respectful organization who published it for verification. This is a time-consuming process and is controlled by human, which is not reliable in some case. This is where our solution comes in. To avoid the unnecessary loss of time we can perform this verification process very quickly in a more secured way at anywhere at any time by just using this web application. There is three part in this project

1. Storing the Main Document
2. Verify the given Document
3. Download a particular Document

To verify any digital document, we need the main document and match it with the given.

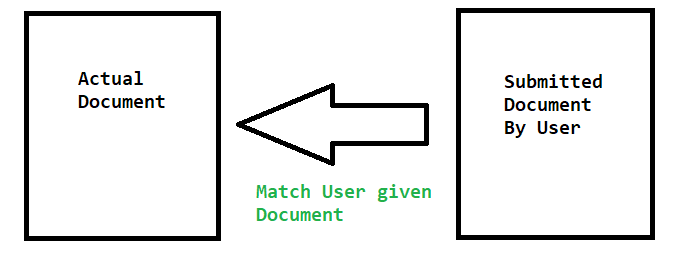


Figure 1.1.1: Document Verify

But storing and matching the whole document bit by bit will use up a lot of storage and processing power. We checked that problem using generating Hash for a file using SHA 256 Hashing Method and for a secured storage system used Blockchain.

Then to check any documents validity verification process can be used and to do this the document must be uploaded before. If a document is trying to be verified but it is not yet uploaded to the blockchain an error will be shown, also if the document’s hash value does not match with the uploaded document’s hash value that means the document has been manipulated without the concern of its owner which is illegal so in that case an error message will also be shown in the screen to let the user know about error. Now if a document is verified successfully then it’s corresponding hash value will be given to the user for the purpose of downloading it in local storage to use.

So this whole process is done by using some extra-ordinary advance technologies. Ethereum blockchain is the main base of the system which stores all the data. To get Ethereum properly working we also need some important dependencies like **Metamask, Web3.js** etc .We will discuss about these dependencies in the next section in details. This whole project is finally deployed to Heroku from which anybody can use this application from their browser with the website link ([**https://doc-block.herokuapp.com/**](https://doc-block.herokuapp.com/)) for file sharing and file verification purpose.

# 2. Used Software & Dependencies

## **2.1 Blockchain**

A blockchain is a decentralized, distributed, and oftentimes public, digital ledger that is used to record transactions across many computers so that any involved record cannot be altered retroactively, without the alteration of all subsequent blocks.

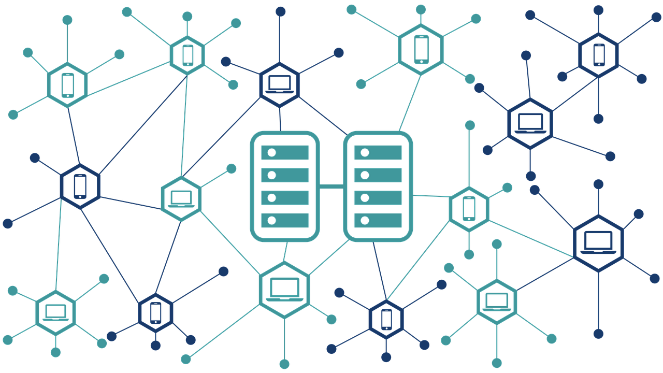


Figure 2.1.1: Blockchain

For use as a distributed ledger, a blockchain is typically managed by a peer-to-peer network collectively adhering to a protocol for inter-node communication and validating new blocks.

Once recorded, the data in any given block cannot be altered retroactively without alteration of all subsequent blocks, which requires consensus of the network majority.This allows the participants to verify and audit transactions independently and relatively inexpensively.

Blocks hold batches of valid transactions that are hashed and encoded into a Merkle tree. Each block includes the cryptographic hash of the prior block in the blockchain, linking the two. The linked blocks form a chain. This iterative process confirms the integrity of the previous block, all the way back to the original genesis block.

A blockchain carries no transaction cost. (An infrastructure cost yes, but no transaction cost.) The blockchain is a simple yet ingenious way of passing information from A to B in a fully automated and safe manner. One party to a transaction initiates the process by creating a block. This block is verified by thousands, perhaps millions of computers distributed around the net. The verified block is added to a chain, which is stored across the net, creating not just a unique record, but a unique record with a unique history. Falsifying a single record would mean falsifying the entire chain in millions of instances. That is virtually impossible. Bitcoin uses this model for monetary transactions, but it can be deployed in many other ways.

The blockchain network has no central authority — it is the very definition of a democratized system. Information held on a blockchain exists as a shared — and continually reconciled — database. This is a way of using the network that has obvious benefits. The blockchain database isn’t stored in any single location, meaning the records it keeps are truly public and easily verifiable. No centralized version of this information exists for a hacker to corrupt. Hosted by millions of computers simultaneously, its data is accessible to anyone on the internet.

There are three main properties of Blockchain Technology which have helped it gain widespread acclaim are as follows:

1. Decentralization
2. Transparency
3. Immutability

## **2.2 Ethereum**

Ethereum is a distributed public block chain network that focuses on running programming code of any decentralized application. It is an open source, public, blockchain-based distributed computing platform and operating system featuring smart contract (scripting) functionality. More simply, it is a platform for sharing information across the globe that cannot be manipulated or changed.

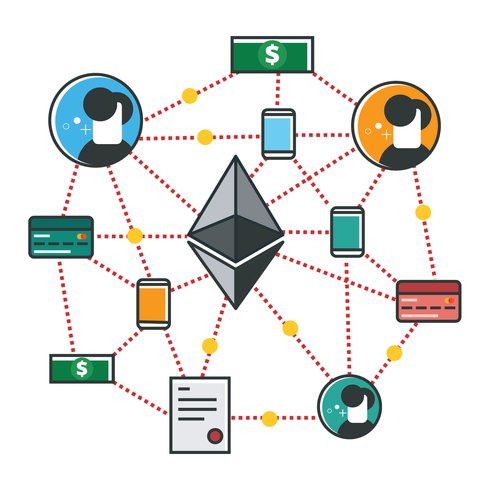


Figure 2.2.1: Ethereum

Ether is a cryptocurrency generated by the Ethereum platform and used to compensate mining nodes for computations performed. It is a decentralized digital currency, also known as ETH. In addition to being a tradable cryptocurrency, ether powers the Ethereum network by paying for transaction fees and computational services. Each Ethereum account has an ether balance and ether may be transferred from one account to another.

When anybody send ether or do anything else on the Ethereum blockchain, he must pay miners for the computation of that transaction.in his blockchain wallet, we'll set this as a fixed fee for you.

Ethereum provides a decentralized virtual machine, the Ethereum Virtual Machine (EVM), which can execute scripts using an international network of public nodes. The virtual machine's instruction set, in contrast to others like Bitcoin Script, is Turing-complete. "Gas", an internal transaction pricing mechanism, is used to mitigate spam and allocate resources on the network.

## **2.3 Solidity**

Solidity is an object-oriented programming language for writing [smart contracts](https://en.wikipedia.org/wiki/Smart_contract). It is used for implementing smart contracts on various blockchain platforms, most notably, Ethereum.

Solidity is a [statically-typed](https://en.wikipedia.org/wiki/Statically_typed_programming_language) programming language designed for developing smart contracts that run on the EVM. Solidity is compiled to [bytecode](https://en.wikipedia.org/wiki/Bytecode) that is executable on the EVM. With Solidity, developers are able to write applications that implement self-enforcing business logic embodied in smart contracts, leaving a non-reputable and authoritative record of transactions. Writing smart contracts in smart contract specific languages such as Solidity is referred to as easy (ostensibly for those who already have programming skills).

## **2.4 Infura**

To avoid running a Ethereum node in our PC to run our system, we used Infura.

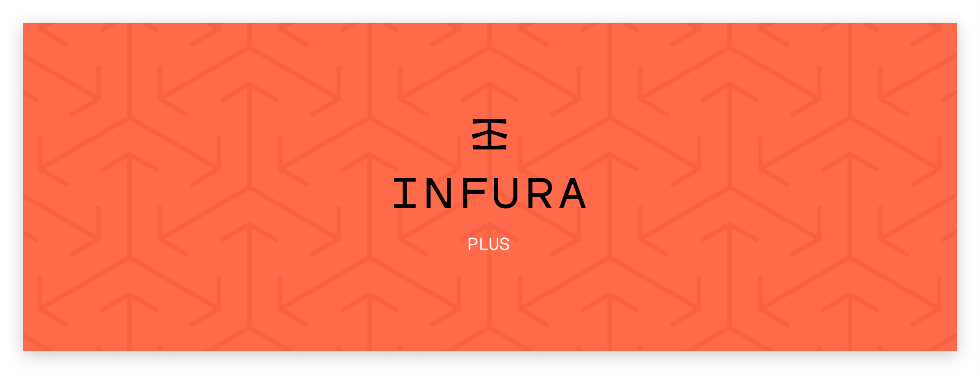


Figure 2.4.1: Infura

**Infura** is a hosted Ethereum node cluster that lets users run application without requiring them to set up their own Ethereum node or wallet.

## **2.5 IPFS**

The Inter Planetary File System (IPFS) is a protocol and peer-to-peer network for storing and sharing data in a distributed file system. IPFS uses content-addressing to uniquely identify each file in a global namespace connecting all computing devices.

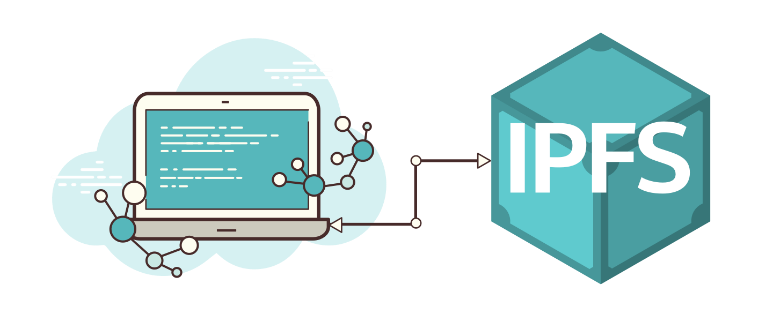


Figure 2.5.1: Ipfs

IPFS allows users to not only receive but host content, in a similar manner to BitTorrent. As opposed to a centrally located server, IPFS is built around a decentralized system of user-operators who hold a portion of the overall data, creating a resilient system of file storage and sharing. Any user in the network can serve a file by its content address, and other peers in the network can find and request that content from any node who has it using a distributed hash table (DHT).

# 3. Methodology to Develop the System

## **3.1 Hashing Method SHA-256**

Storing the actual file in the database will occur in need of a large storage. So, we need some way to map a document uniquely and precisely. For this mapping we can use Hashing. We used SHA-256 hashing method. SHA-256 operates in the manner of MD4, MD5, and SHA-1. The message to be hashed is first

(1) Padded with its length in such a way that the result is a multiple of 512 bits long, and then

(2) Parsed into 512-bit message blocks M(1) ,M(2) ,…..,M(N) .

The message blocks are processed one at a time: Beginning with a fixed initial hash value H(0) , sequentially compute

H(i) = H(i-1) + CM(i) (H(i-1))

Where C is the SHA-256 compression function and + means word-wise mod 232 addition. H(N) is the hash of M.

The SHA-256 compression function operates on a 512-bit message block and a 256- bit intermediate hash value. It is essentially a 256-bit block cipher algorithm which encrypts the intermediate hash value using the message block as key.

Six logical functions are used in SHA-256. Each of these functions operates on 32-bit words and produces a 32-bit word as output. Each function is defined as follows:

Expanded message blocks W0W1………..W63 are computed as follows via the

SHA-256 message schedule:

Wj = Mj(i) for j = 0,1…15, and

For j = 16 to 63

{

}

Diagrams:

The SHA 256 compression function in pictured below:

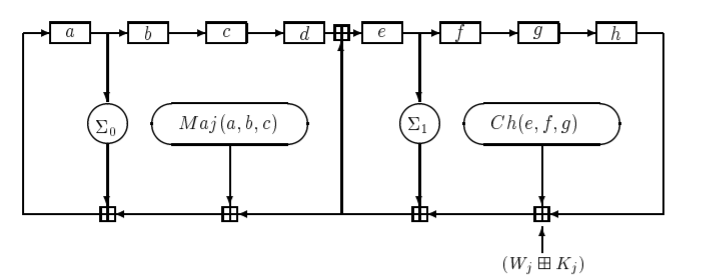


Figure 3.1.1: Jth internal step of the SHA – 256 Compression function C

Where the symbol denotes mod addition.

The message schedule can be drawn as follows:

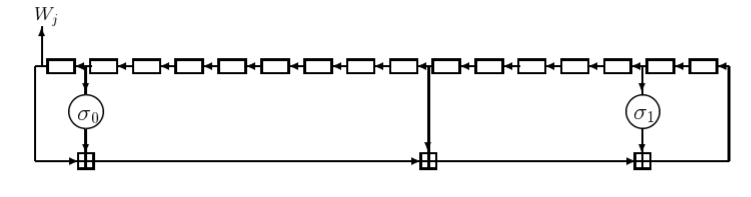


Figure 3.1.2: SHA -256 message schedule.

The probability of two hash collision with each other is,

This means that the probability is negligible as long as we have significantly less than 2128 values. That's the case for any realistic amount of data, so an unbroken 256-bit hash is good enough and there is no need to upgrade to 512 bits.

Now we can generalize the verification procedure as this,

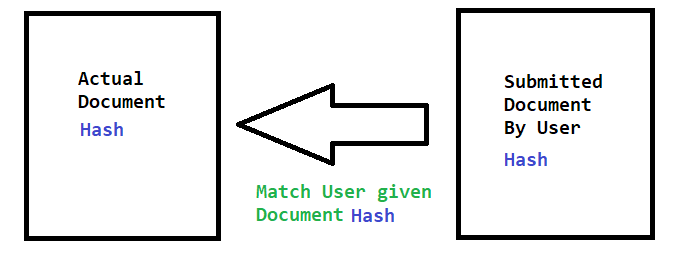


Figure 3.1.3: Hash Matching.

As a result, we don’t need to save whole document in the database. Just the Hash and match it with user submitted document.

## **3.2 Security in Decentralized Database**

A centralized database server is controlled by single authorization, thus backdooring changes can be done using security flaws, or the database maintainer. To make the database more secure we used a decentralized way to store this hashes, Ethereum Blockchain.

Ethereum is a global, open-source platform for decentralized applications.

On Ethereum, you can write code that controls digital value, runs exactly as programmed, and is accessible anywhere in the world. Any program that runs on the Ethereum Virtual Machine (EVM) is commonly referred to as a “smart contract”. We have used Solidity language to create the smart contract. Main purpose of our contract is to store hash value of the file and organizer name. Then match the hash string with the user query. As adding any value to blockchain cost gas, average cost of adding a value if 0.00011 Ether, which is approximately cost of a phone call.

### Why it’s more secured than storing the data in a typical centralized database?

A blockchain is a public ledger of information collected through a network that sits on top of the internet.

Blockchain technology is not a company, nor is it an app, but rather an entirely new way of documenting data on the internet

Blockchains are secured through a variety of mechanisms that include advanced cryptographic techniques and mathematical models of behavior and decision-making. Blockchain technology is the underlying structure of most cryptocurrency systems and is what prevents this kind of digital money from being duplicated or destroyed.

The information recorded on a blockchain can take on any form, whether it be denoting a transfer of money, ownership, a transaction, someone's identity, an agreement between two parties, or even how much electricity a lightbulb has used. However, to do so requires a confirmation from several of devices, such as computers, on the network. Once an agreement, otherwise known as a consensus, is reached between these devices to store something on a blockchain it is unquestionably there, it cannot be disputed, removed or altered, without the knowledge and permission of those who made that record, as well as the wider community.

# 4. Implementation:

## **4.1 Designing the Contract**

Our contract will have set / get options for Hash Value which will store Document Added name and Date of adding.

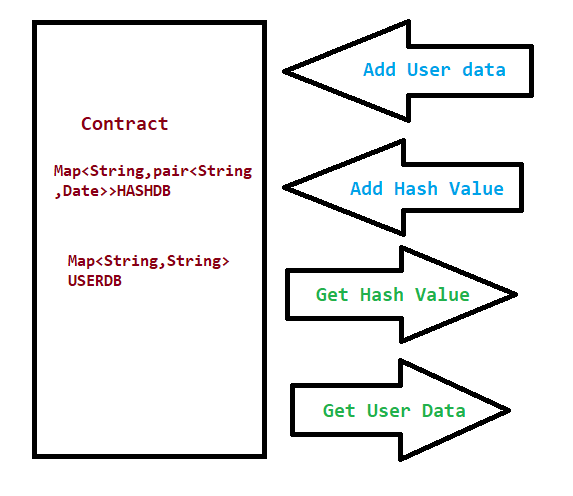
And another DB for User. Generally, it will be look like this:

Figure 4.1.1: Smart Contract

Now, in our contract we have,

**mapping (string => uint256) documents; // store date and time against file hash**

**mapping (string => string) adderkey; // store adder key against file hash**

**mapping (string => string) addername; //store added name againt adder key**

### Add a document

**function add(string memory \_hash,string memory x) public {**

**if(documents[\_hash]>0)return;**

**documents[\_hash] = block.timestamp;**

**adderkey[\_hash] = x;**

**}**

### Verify a document

**function verify\_doc(string memory \_hash) public view returns (uint256 dateAdded) {**

**return documents[\_hash];**

**}**

### Get document adder info

**function getadderkey(string memory \_hash) public view returns (string memory name){**

**return adderkey[\_hash];}**

**function getaddername(string memory \_pubkey) public view returns(string memory name)  {**

**return addername[\_pubkey];**

**}**

### Set user name

**function verify\_name(string memory \_pubkey) public view returns(uint ok) {**

**bytes memory mem = bytes(addername[\_pubkey]);**

**if(mem.length>0){**

**return 1;**

**}**

**return 0;**

**}**

**function setaddername(string memory \_pubkey,string memory name) public {**

**addername[\_pubkey] = name;**

**}**

## **4.2 Creating the Web Application**

We used Nodejs to create our website. Our website will basically need to show this:

1. User Info

2. Document Verification Info

3. Add Document to blockchain

4. Messages

First our node app will connect to Ethereum blockchain. We will use Metamask extension for browser to connect to Ethereum blockchain and web3.js to interact with our contract.

### Check Metamask and web3

**//check if Ethereum web3 is available**

**if (window.ethereum)**

**try {**

**await window.ethereum.enable();**

**} catch (err) {**

**return App.showError(err.message.toString() + err.stack.toString());**

**}**

**//check metamask is inabled**

**if (typeof web3 === "undefined") {**

**return App.showError(err.toString());**

**}**

Then we will fetch user info if available otherwise get and set user info from user.

### Fetch and set User Info

**//init contract**

**//get name using public key**

**contractx.getaddername(App.account.toString(), function (err, result) {**

**if (result === null || result === "") { //name is not set**

**get\_name.show(); //signup**

**} else {**

**App.name = result.toString(); //signed up, set name**

**$("#accountName").html(App.name);**

**}**

**}**

**});**

**//handle errors**

### Set User Name

**//init contract**

**// set name**

**contractx.setaddername(App.account, s, function (err, result) {**

**App.name = s;**

**$("#accountName").html("Account Name: " + App.name);**

**content.show();**

**});**

**//handle erros**

## **4.3 IPFS for Upload/Download**

Storing a document in a blockchain will cost a lot and storing in Online Database will cost security issues. This is where IPFS comes in.

The Inter Planetary File System is a peer-to-peer hypermedia protocol designed to make the web faster, safer, and more open. IPFS has a p2p protocol to store and share file. We used it to creating the file sharing service for the users.

To Upload a file in IPFS we will create a IPFS Node first and then push it

Similarly, to download a file we will create a IPFS node and then get the IPFSHASH from user and download it.

# 5. Application Functionality

## **5.1 Verify Document**

A general way of verify a document can be diagramed as this:

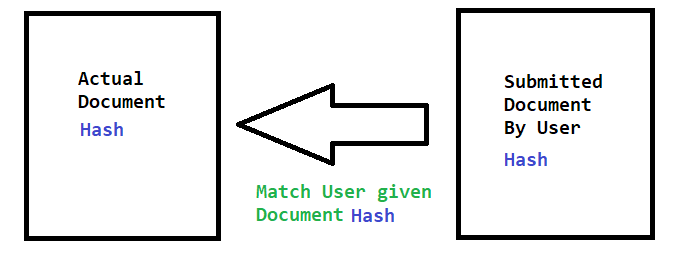


Figure 5.1.1: Verification

We need the main document that was published from the authorizer. Then we match the user given document to get result. Any change even an increase of space in the document should generate false document warning.

### Verifying a document

**//init contract**

**contractx.verify\_doc(s, function (err, result) {**

**let isReleased = result.toNumber();**

**if (isReleased > 0) { //released**

**contractx.getadderkey(s, function (err, result) {**

**//get adder key and name**

**let ac = result.toString();**

**contractx.getaddername(ac, function (err, result) {**

**});**

**});**

**}**

**});**

**// handle errors**

**}**

### Generate SHA-256 hash

**const file = event.target.files[0];**

**const reader = new window.FileReader();**

**reader.readAsDataURL(file);**

**reader.onloadend = () => {**

**var x = reader.result.toString();**

**App.buffer = sha256(x);**

**App.buffer2 = x;**

**};**

Now our website is ready for verify documents. But we wanted to add another service file Downloading/uploading

## **5.2 Upload Document**

### Adding a Document

**//after adding file to blockchain**

**App.node.add(App.buffer2, function (errx, resipfs) {**

**return App.showInfo('file added :: ' + result + '<br><h7 class="text-danger">Please save this IPFS HASH This will be shown only for one time</h7><br>IPFS HASH :: ' + resipfs[0].hash);**

**//Handle errors**

## **5.3 Download a Verified Document**

**var filebuffer = await App.node.cat(hashtext);**

**var stringval = filebuffer.toString();**

**//decode and download file**

We will store and give the ipfs hash to user when adding a document. This IPFS Hash won’t be save on others network and will be shown only one time to users for security purpose.

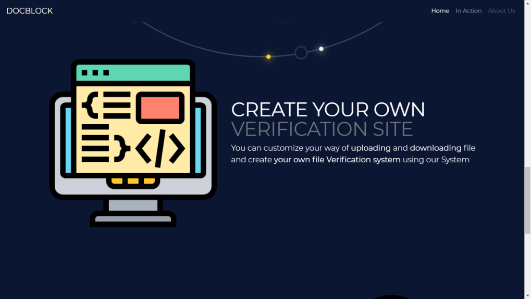
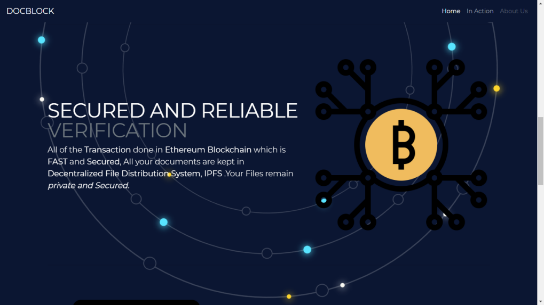
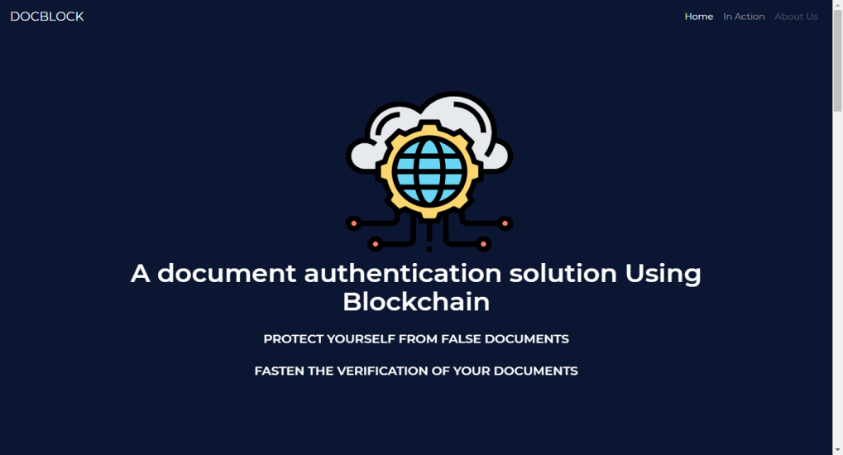
# 6. Result Analysis

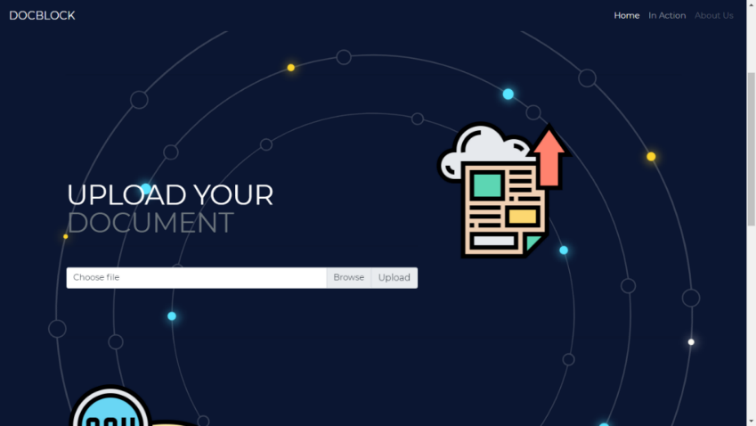
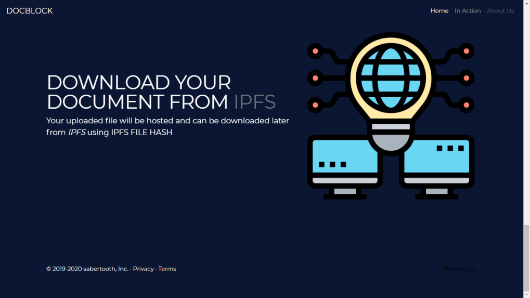
As the File sharing service has been added now we then deployed it to Heroku and our website is Live! [[Github](https://github.com/sabertooth9/dapp-docblock/)]

[**https://doc-block.herokuapp.com/**](https://doc-block.herokuapp.com/)

Website’s user interface was designed manually keeping the concern in head of behaving user friendly so that a new user who doesn’t have any knowledge about Ethereum or blockchain can easily operate this website for any professional work or something. The website does not have a lot of content neither have any useless info or unexpected behavior. It is very simple and decent to use for anybody. Here is some still pictures of our website:

## Website Demo





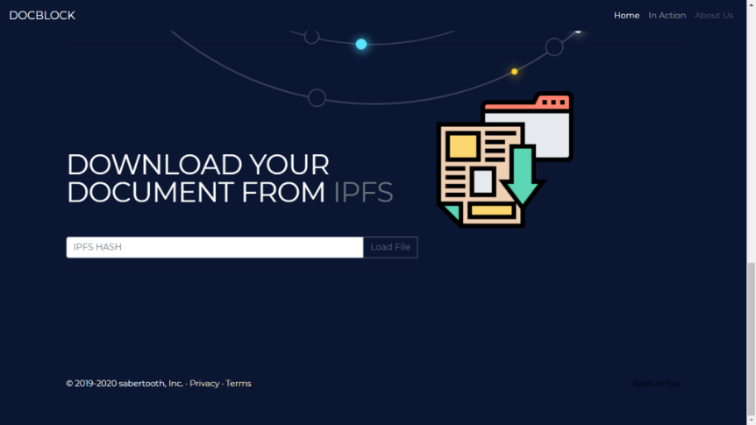


Figure 6.1.1: Screenshots

As our website is live , we added a file , you can download it from ipfs using this ipfs-hash: QmWBxyHwyBTmoy83vFw8srfwG1mVQfM8KggmzqV7X6siWU

This file generated sha-256 Hash is 4ff1aaef4f879001104c5d12cafaa6394561a37a99610038b7232498d6e39f7a

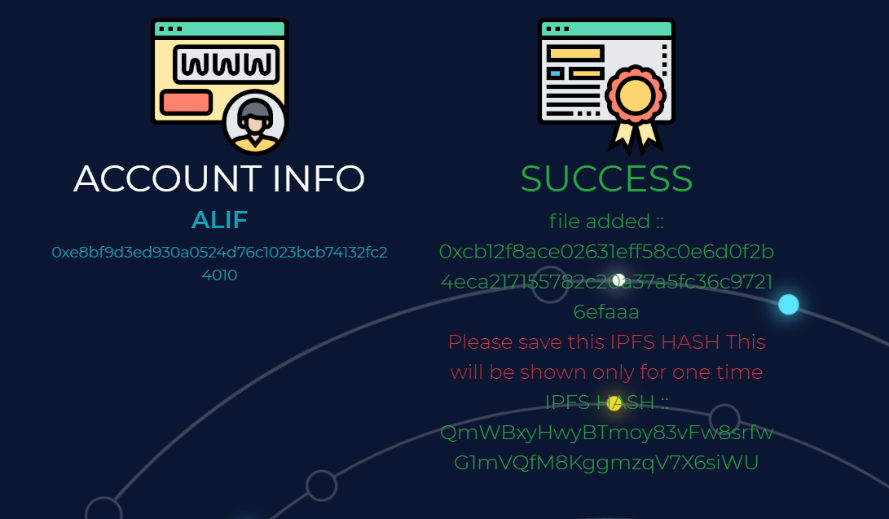


Figure 6.1.2: Adding a document

After Adding we can download and verify this file. Typing the IPFS-Hash and pressing the load button downloads the file from IPFS



Figure 6.1.3: Download a document

We can verify this file loading it in Verify Section.

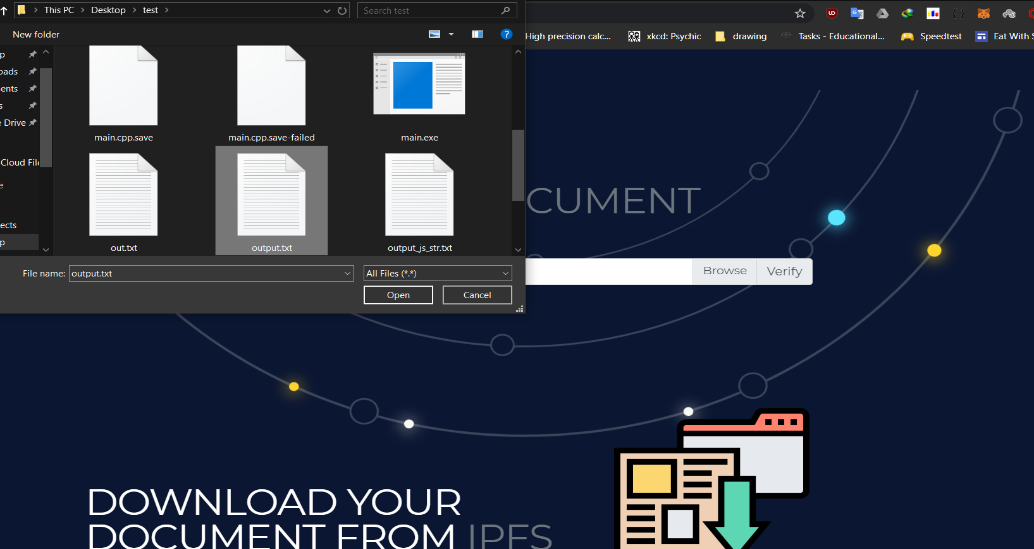


Figure 6.1.4: Uploading a file for verification

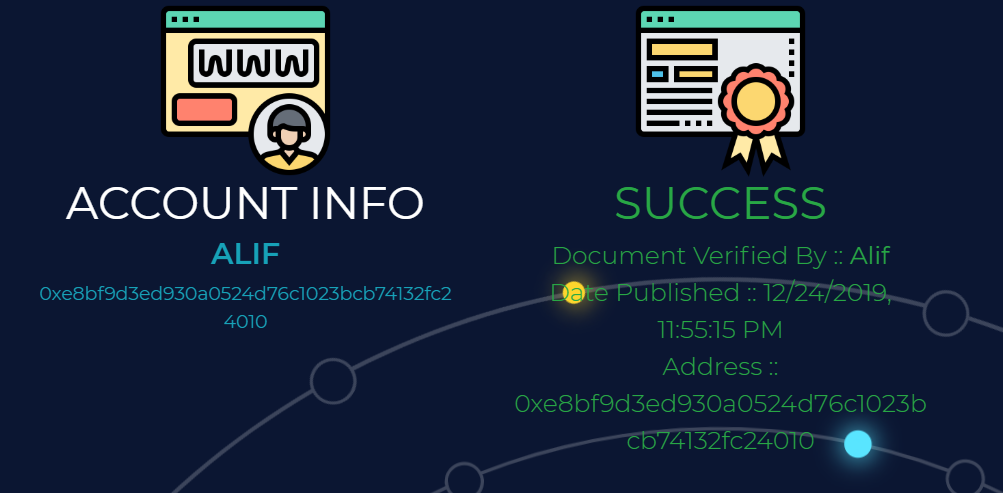
After hitting the Verify Button : 

Figure 6.1.5: Verification Info Success

Then we added a single space in the same file and sha256 hash is:

a6a68ed82691ee2eeb94e79c3a28d134f447a917f6fa4c050ba3e046a58ee784

We can already see that what a single space did to the sha256. Now if we hit verify Verify button it will give us error that this file was not verified by anyone



Figure 6.1.6: Verification Info Error

The probability of two hash collision with each other is,

This means that the probability is negligible as long as we have significantly less that, 2128 values. That's the case for any realistic amount of data, so an unbroken 256-bit hash is good enough and there is no need to upgrade to 512 bits.

We can see that as SHA256 is one-way and collision-resistant and our blockchain and IPFS is working as expected.

# 7. Limitations

Although we created this project very carefully and deployed it nicely it has some limitations.

1. Required Crypto to Use
2. Required Internet Connection
3. Small amount of crypto (Equivalent to calling someone for a minute) is required to Add a document / Sign Up

# 8.Conclusion

The main idea of this project is to create a system which will play an important role in checking the validity of a file, document, certificate, land/property/asset document more accurately in short period of time.

SHA-256 is one-way and collision-resistant. We used some well known and secured methods Blockchain to develop the System. We thoroughly explained why we choose this way to verify a document and why it’s a reliable and secured system. Then we explained about the SHA-256 Cryptography and how we implemented the full system and deployed it.

The whole process is very secure that it is quite impossible for hackers to manipulate any document. Thus, this project stands strongly in the basis of strong security. Users never have to worry about their data to be deleted or corrupted and they can also easily access their documents whenever they want.

The main purpose of the project was fulfilled successfully. The project is user friendly and less time consuming but more secure. We can work on this project more to add more unique features. Hopefully this application will play a vital role in digital file verification system.

# 9.Future Plans

Now any users with crypto, can access /add file to our network. But with some modification we can make the “Adding Document” particular user only and personalized website for the that organization and other users will able to verify without any crypto.

This will be more flexible and usable for an organizer and their userbase.

# 10.References

1. <https://ethereum.org/>

2. <https://ipfs.io/>

3. <https://www.dappuniversity.com/>

4. <https://dashboard.heroku.com/login>

5. <https://metamask.io/>

6. <https://infura.io/dashboard>

7. CSE 2101 Object Oriented Programming

8. CSE 2105 Data Structures and Algorithms

9. CSE 2201 Algorithm Analysis and Design

10. CSE 3100 Web Programming Laboratory

11. <https://www.movable-type.co.uk/scripts/sha256.html>