Decentralized Cloud Storage with Searchable Encryption

4/12/2019

Dr. John Yeun
Tejasvi Mehra
Student ID: 3035239600
The University of Hong Kong
Department of Computer Science
Abstract

This project aims to utilize the power of decentralized storage along with added encryption to provide an extremely secure cloud storage application. A big part of the application is searchable encryption which is used to search for keywords in encrypted documents without decrypting any document. The main reason for working on this application is the increased security, reduced cost and data safety options it can provide when compared to a standard centralized storage application such as Dropbox. The project is completed and works for a limited file size and type at the moment. The application is paid and uses Ether as currency.
Acknowledgment

I will like to thank our project supervisor Dr. John Yeun for providing us with the support and giving us the opportunity to develop this application under his tutelage. I will also like to thank my team for being so supportive and optimistic even in the face of numerous challenges and keeping us on schedule.
Table of Contents

Abstract .......................................................................................................................... 1
Acknowledgment............................................................................................................ 2
List of Figures .................................................................................................................... 4
List of Tables .................................................................................................................... 4
Abbreviations .................................................................................................................. 4
1. Introduction ............................................................................................................... 7
   1.1. Background .......................................................................................................... 8
       1.1.1. Decentralized Technology .......................................................................... 8
       1.1.2. Blockchain .................................................................................................. 9
       1.1.3. Cryptography .............................................................................................. 11
   1.2. Motivation ............................................................................................................ 15
       1.2.1. Security ........................................................................................................ 15
       1.2.2. Data Redundancy ....................................................................................... 16
       1.2.3. Cost Efficiency .......................................................................................... 17
   1.3. Deliverables ......................................................................................................... 17
   1.4. Scope ................................................................................................................... 18
   1.5. Outline ................................................................................................................. 18
2. Methodology ............................................................................................................ 19
   2.1. Technology Stack ............................................................................................... 19
   2.2. UI Design ........................................................................................................... 23
   2.3. Framework ......................................................................................................... 24
       2.3.1. Development Process ............................................................................... 24
       2.3.2. Components ............................................................................................. 25
       2.3.3. Workflow Steps ....................................................................................... 28
       2.3.4. Node Modules .......................................................................................... 33
3. Searchable Encryption .............................................................................................. 36
   3.1. Security in Searchable Encryption ...................................................................... 37
   3.2. Workflow ............................................................................................................ 38
4. Results ....................................................................................................................... 44
5. Challenges .................................................................................................................. 45
6. Conclusion .................................................................................................................. 47
7. References .................................................................................................................. 48
List of Figures

Figure 1 – Centralized vs Decentralized Network ........................................... 9
Figure 2 – Working of Blockchain ..................................................................... 11
Figure 3 – SubBytes Transformation .................................................................. 13
Figure 4 – ShirtRows Transposition ................................................................... 13
Figure 5 – MixColumn Transformation ................................................................. 13
Figure 6 – AddRoundKey .................................................................................. 14
Figure 7 – One iteration within the SHA-1 compression function ....................... 15
Figure 8 – Welcome Screen .............................................................................. 29
Figure 9 – Login Page ....................................................................................... 29
Figure 10 – Registration Page ............................................................................ 30
Figure 11 – Upload Page .................................................................................... 31
Figure 12 – Download Page ............................................................................... 32
Figure 13 – Decrypt Page .................................................................................. 32
Figure 14 – Working of Searchable Encryption .................................................. 38
List of Tables

Table 1 – Table for mapping all keywords in a document .................. 40
Table 2 – Inverted index table mapping keyword to all documents ...... 40
Table 3 – Encrypted Document with Document ID .......................... 41
Table 4 – Hashed keyword with Document ID ................................. 42
Abbreviations

IPFS – Interplanetary File System
APIs – Application Program Interface
AES – Advanced Encryption Standard
SHA-1 – Secure Hash Algorithm
DES – Data Encryption Standard
NSA – National Security Agency
RSA – Rivest–Shamir–Adleman
GB – GigaByte
MB – MegaByte
REST – REpresentational State Transfer
P2P – Peer to Peer
NPM – Node Package Manager
I/O – Input/ Output
HTTP – Hyper Text Transfer Protocol
JSON – JavaScript Object Notation
dAPP – Decentralized Application
MEW – MyEtherWallet
UI – User Interface
URL – Uniform Resource Locator
HTML – Hypertext Markup Language
OAuth – Open Authorization
1. Introduction

Blockchain Technology is gaining a lot of attention these days to the powerful and new capabilities it has to offer. Developed by Satoshi Nakamoto in 2008, Blockchain is the technology used to create and manage a digital ledger which is available simultaneously to everyone who is a part of the chain [1]. A parallel can be drawn between a Google doc and Blockchain ledger to better understand the working of the ledger. A Google doc provides the same content to everyone who has access to the document. Any subsequent change is updated in real time for everyone. Similar to Google doc, Blockchain is a network of information available to everyone with any new changes or additions reflected in everyone’s personal ledger at the same time. This project aims to utilize blockchain to provide a secure way to make transactions for a decentralized cloud storage application which is built using APIs from IPFS along with added encryption to secure the files stored.
1.1. Background

1.1.1. Decentralized Technology

A traditional centralized network more commonly known as “server-client network” depends on the server. In the event of server failure, the entire network fails, leading to a loss of information for all the parties involved. In a decentralized network, the control is distributed across its entirety. A decentralized network has no central point of control and every “node” has a copy of the entire information set. Decentralized cloud storage is a type of network where the data is not stored in a centralized data center, but, is instead broken or “sharded” down and distributed to multiple nodes on the network. A common example of decentralized storage is the Torrent. Torrent work on the principle of decentralization where files are not stored centrally but are available to multiple nodes across the network. A downloader connects itself to a “swarm” (a group of computers uploading and downloading the particular file) to transfer data between each other. The downloader shares his IP with the swarm and starts downloading the file in little pieces. At the same time, the downloader also uploads the parts which have been downloaded onto his machine, thus speeding up the
download speed of others in the swarm. In this manner, there is no central server and instead, the files are hosted by multiple users in a decentralized manner [2]. Even if one of the nodes is removed, the file is available through the other nodes.

Figure 1 – Centralized vs Decentralized Network.
A centralized network has a single point of control and thus a central point of failure. A decentralized network has its control distributed across the network with no single point of failure [3]

1.1.2. Blockchain

A Blockchain can be understood simply as a series of time-stamped, immutable records of data that are controlled by a cluster of computers, not under the control of any single entity. Blockchain is an ever-growing list of records. Each data block (i.e. “block”) is secured and bound to others through cryptography (i.e. “chain”) [4]. This is a direct application of decentralized systems as there is no authority who controls the entire application. It provides a way to make peer-to-peer
transactions. A user in the Blockchain network requests a transaction. A transaction can be any form of information exchange on the Blockchain network but is used for mostly currency. The transaction is then broadcasted to the Blockchain peer to peer network consisting of multiple computers called “nodes”. The nodes then check the correctness of the transaction through known algorithms. Once verified by a majority of nodes, the transaction is combined with multiple other records to create a new data block. Since the block is approved by rule of majority, it is a favorable approach compared to a centralized transaction where the entity in charge of the transaction may be biased. This block is added to the existing chain in a manner that is irreversible. The transaction is finally marked complete.
Figure 2 - Working of Blockchain. A transaction is created by a user. It is then approved by the nodes in the network. After approval the block is added to the chain.

1.1.3. Cryptography

Cryptography is a field of computer science which deals with creating and using technology to prevent intruders from accessing private information including data confidentiality, data integrity, authentication, and non-repudiation. (Introduction to Modern Cryptography.) It deals with a “plaintext” which is encrypted with an encryption algorithm to obtain a “ciphertext” which is then sent across an unsafe network to prevent intruders from accessing the private information.

This project utilizes many two major cryptographic techniques which are discussed in detail below.
1.1.3.1. AES

AES is a block cipher with a block size of 128 bits. It is a type of symmetric key encryption i.e. the same key is used for encryption and decryption. AES is a based-on substitution-permutation network and is a subset of Rijndael block cipher. It is considered as one of the top ciphers of the world at this moment and has not been cracked yet. It has superseded DES and is in use worldwide. AES has three variants for key sizes – 128, 192 and 256 bits. AES-256 is used in the project to encrypt the uploaded files. For AES-256, 14 transformation rounds take place on the plaintext to obtain the ciphertext [5].

- **Key Expansion** – AES uses a key schedule to expand the initial key into different round keys which are used in the subsequent rounds. Each round needs a separate 128-bit round key plus one key used during initial key addition.

- **Initial round key addition** – Before any round-based processing begins, the first key is added to the block using bitwise XOR.

- **Round 1 to 14** – Four steps namely SubBytes, ShiftRows, MixColumns, and AddRoundKey are carried out in each round except the last round in which AddRoundKey is skipped. Each step is explained in detail below.
⇒ **SubBytes**: Using a lookup table, all the bytes are substituted non-linearly with a different byte.

![SubBytes Transformation](image)

**Figure 3 – SubBytes Transformation [6]**

⇒ **ShiftRows**: Transposition takes place by cyclically shifting the last three rows of the state by a certain number of steps.

![ShiftRows Transposition](image)

**Figure 4 – ShiftRows Transposition [6]**

⇒ **MixColumns**: The four bytes in the column of the state are combined in a linear mixing state.

![MixColumns Transformation](image)

**Figure 5 – MixColumn Transformation [6]**
$\Rightarrow$ **AddRoundKey**: Each byte in the state is bitwise XORed with the round key of that round.

![AddRoundKey Diagram](image.png)

**Figure 6 – AddRoundKey [6]**

1.1.3.2. SHA – 1

SHA-1 is a hashing algorithm which creates a 160-bit or 20-byte output from any given input. It was developed by the NSA and is an information processing standard. It is a one-way function meaning the original input cannot be obtained back from the given hash. It is fast to compute and is used in the program to hash the values of the tags before sending them to the server. For any small change in input the output hash changes completely thus it is an important tool for checking data integrity. It is also collision resistant meaning that any two inputs will not give the same output hash.
1.2. Motivation

1.2.1. Security

Traditional cloud storage applications such as Dropbox, iCloud etc. store the data in data centers which are susceptible to data leaks. These data centers are targeted by people for improper and illegal use. Encrypted data can still be leaked. Over 500 pictures
were leaked from iCloud in 2014 [8] and Dropbox lost the credentials of over 68 million of its users [9]. Thus, there is a need for a more secure form of cloud storage which can be achieved by decentralization. Data stored in a decentralized network is secured through AES in this project making it much harder to compromise. By its very nature, it is a peer to peer application thus, any hacker will have to gain access to numerous devices at the same time to access a single piece of information which is not practical nor possible.

1.2.2. Data Redundancy

The data centers in a cloud storage application store all the data. There may be multiple data centers across the world but the information belonging to any user is not stored in more than one location. Thus, during physical damage to a data center, all the data is lost and is irretrievable. In a decentralized cloud storage, there are no data centers and instead, the data is stored in multiple nodes. There are copies of the same data in multiple nodes to ensure data redundancy. Thus, during physical damage to a data node, the information is still safe in another node in the network.
1.2.3. Cost Efficiency

Since this does not require any data centers to be set up and will be using existing space available with day to day users, utilizing this type of decentralized storage will reduce cloud storage costs by up to 100 times. Currently, Dropbox offers 100 GB of storage for $99. A decentralized cloud storage company called Storj which based on a similar idea provides the same storage at the cost of $1.47 including backup copies and charges $0.49 for retrieval making the total cost of storage $1.96 per user [10].

1.3. Deliverables

The final goal of this project was to develop a web-based application with a graphical interface to support the features developed. For file uploaders, hereby referred to as simply “uploaders”, the application will allow the user to upload a file to the decentralized data storage and receive a confirmation. The uploaded file will be encrypted before being stored to add an additional layer of security. Moreover, the user also adds tags related to the uploaded file to enable tag-based searching. The currency Ether is used to deduct payment from the uploaders for each file upload.

For file downloaders, the application allows them to download
the file by searching based on the file name as well as searching by tags. Each file is uploaded with the hashed combination of filename and username as a default tag to allow the user to directly retrieve this file. The encrypted file is fetched from storage and decrypted and returned to the user.

1.4. Scope

This project has enormous potential in terms of security and encryption, cost and data redundancy due to the underlying technologies used. This project will allow uploaders to store documents in a secure decentralized storage for virtually no cost.

1.5. Outline

This report seeks to familiarize the reader with the different technologies utilized in this project and framework which includes the development process, the components and the workflow steps. Searchable encryption is discussed in detail due to the big part it plays in the application. Finally, it takes note of the results, challenges faced during the development cycle and the limitations of the project.
2. Methodology

2.1. Technology Stack

1. *Ethereum Blockchain* –

This project uses the Ethereum Blockchain technology developed by The Ethereum Foundation.

**Justification:** Ethereum is extremely powerful and versatile with services which allow new code to be deployed independently as isolated programs allowing an easy to use test environment for the pilot program. Moreover, it will allow the usage of Ether as a cryptocurrency to be used for payment. Ethereum has an active community who is extremely capable and helpful compared to other Blockchain communities which will allow faster and more efficient development.

2. *Solidity* –

This project will use Solidity v0.4.25. Solidity is derived from high-level programming languages such as Python and JavaScript to help create a stable working environment.

**Justification:** Solidity is a proprietary programming language for Ethereum Blockchain used to create and
implement “Smart Contracts”. A Smart Contract for a blockchain is used to create a virtual agreement like a paper contract for agreeing to certain conditions before implementing payment. For this project, a smart contract is used to pay for the file uploaded by a user to the decentralized storage.

3. **IPFS** –

This project uses IPFS to store the files. IPFS is a decentralized cloud-based storing company which uses distributed hash tables to store the file data for users. IPFS can be understood as a single Torrent swarm who exchange information objects within a GitHub repository. Each file and block have a unique fingerprint which is used to identify it.

**Justification:** It is open source and has a REST API which is utilized to implement the basic workings of the application like adding files to the decentralized storage, retrieving files, file splitting or “sharding”. This is done to reduce the need to implement these features from scratch. Each connected node only stores information which it wants to along with the identification information [11]. The
main advantage of using IPFS over other applications such as Storj and FileCoin is that IPFS is relatively bug-free with good documentation available.

4. NodeJS –
NodeJS is an asynchronous event-driven JavaScript runtime. It executes JavaScript code outside the browser as a standalone application on a machine.

**Justification:** NodeJS is open source and extremely scalable. NPM is the largest collection of open source libraries in the world. A lot of the libraries used are directly imported from NPM. It also has a non-blocking I/O model i.e. it can handle a lot of concurrent requests at the same time without multithreading. It has inbuilt capabilities for handling HTTP requests making it extremely suitable for the project.

5. MongoDB –
MongoDB is a NoSQL based database system which stores data in JSON-like document. The application uses Mongoose as the MongoDB wrapper to interact with the
database. MongoDB is used to store the file metadata to boost retrieval time and speed. 

**Justification:** MongoDB is extremely scalable, versatile and works very well along with NodeJS making it the best choice for storing file information.

6. *MetaMask* –

MetaMask is a bridging application which allows the user to connect to a decentralized web through the browser. It is available as an extension in most prominent browsers like Google Chrome, Firefox etc. MetaMask lets the user Ethereum based dAPP on the browser without running a fully functional Ethereum Node. MetaMask is used in conjunction with Web3 (refer to section 2.3.4.5) to execute the smart contract for the application. It comes with a built-in secure identity vault to preserve the user credentials and digital wallet. MetaMask will be responsible for handling the deduction of Ether from the user’s digital wallet after the user uploads the file.

**Justification:** MetaMask is more secure compared to its alternatives such as MEW or Mist. It also provides a user interface to manage transactions. The main reason for
using MetaMask over others is the fact that it is available as a browser extension and thus removes the need to create a wallet for the application.

7. **Visual Studio Code** –

This project will be developed in Microsoft’s Visual Studio Code v1.28 text editor. This is a standard, open source, text editor developed by Microsoft.

**Justification:** Visual Studio Code contains multiple extensions to assist with debugging, deploying and version control. Furthermore, it has extensions to help coding in NodeJS and Solidity with IntelliSense to help with the development process. It is extremely versatile and faster than other text editors which makes it the ideal choice for this project.

### 2.2. UI Design

The User Interface will address all the UI Design Principles as defined by Larry Constantine and Lucy Lockwood in their book *Software for Use: A Practical Guide to the Essential Models and Methods of Usage-Centered Design*. The interface will employ different a page for upload and download. Upload
screen will contain all the basic information such as a file upload button, keywords input box and passphrase input box. Download will have a different page showing two inputs one for searching by tags and other for searching by filename. The UI focuses more on simplicity and will be broken down into multiple pages depending on the use cases of the users. The detailed information for each page is discussed in the workflow section.

2.3. Framework

2.3.1. Development Process

The development process was divided into eight steps as follows:

1. *Set up a Command Line and Graphical Interface* – This was done to familiarize the team with the different aspects of file uploading and downloading. It was completed through NodeJS application.

2. *Deploy a p2p network for the application* – Since this part was extremely complication to build from scratch, it was completed using IPFS internal modules.
3. **Add file encryption for security so that only the owners with passkey can access the file** – To add additional security, encryption was added on the file on the client side. This was done using a JavaScript library Crypto-JS on the client side.

4. **Implement Searchable Encryption** – To search over the stored encrypted files, searchable encryption was implemented from scratch on NodeJS.

5. **Add a redundancy system to ensure files are not lost due to unreliable nodes** - Completed using IPFS modules.

6. **Add file sharding to break large files** – Completed using IPFS modules.

7. **Utilize smart contracts for payments through Ethereum** – Smart contract added on the client side to deduct ether before sending the file to the server using MetaMask.

8. **Add additional features to improve the application**

### 2.4. Components

The application implements four main components—

1. **Store Data** –
   
   Data is stored on storage nodes on the connected IPFS swarm. These nodes belong to individuals who are
called node operators. An encrypted file fragment is downloaded to the local machine of these node operators. Since the file is fragmented, no one node contains the entire file. Moreover, the file is encrypted prior to its fragmentation which adds to its security. There are multiple copies of the same data fragment in different nodes to add data redundancy to ensure that data is not lost in case of node failure as discussed before. When a user uploads a file along with the associated file tags, the file is first uploaded onto a node on the swarm. Once the file is uploaded and its unique file hash is received, the file tags are stored in a MongoDB collection with the file id which is same as the file hash. This is done for searchable encryption and will be discussed in detail in Section 3.

2. *Encrypt/Decrypt Data –*

Encryption is done on the client side of the application. This is to ensure that the server does not have access to any information present in the document. The document is encrypted through the AES algorithm. This is done using the library Crypto-JS and its working is as
discussed in section 1.3.1. The file tags inputted by the user are hashed using the SHA-1. Again, this is done on the client side to ensure that the server does not know the contents of the document. These are sent along with the file and are uploaded to the MongoDB database along with the file hash retrieved from IPFS. During decryption, first, the encrypted file is retrieved from the server. After retrieval, the user’s passkey is needed to decrypt the file. The decryption again happens on the client side for security purposes. Upon decryption, the file is downloaded on the user’s machine.

3. *Retrieve Data* –

Data retrieval is done by the owner or someone who has the owner’s passkey. There are two ways to retrieve the document. First is by filename which is available only for the owner of the document. The other is by searching for the file using the file tags. Any user can retrieve the file using the file tags. The file tags or filename are hashed on the client side using SHA-1 and sent to the server. The server performs searchable encryption on these file tags on the MongoDB database and retrieves
the file hash which matches the most tags. The file is then retrieved from the IPFS swarm using the file hash. IPFS retrieves the shards and collects the file. This file is sent back to the client for decryption.

4. Pay for storage –

After a user attempts to upload a file to the storage network, a smart contract is executed on the client side which deducts some gas from the user through MetaMask. Upon a successful transaction, the file is sent to the server for upload. If the transaction fails for any reason, the file is not uploaded, and an error message is returned to the user.

2.5. Workflow Steps

1. Welcome

The user is initially greeted with a welcome screen with
options to login to the application or register his email to use the application.

![Welcome Screen](image)

*Figure 8 – Welcome Screen*

2. **Login**

The user uses his email and password to login to the application.

![Login Page](image)

*Figure 9 – Login Page*
A new user can register for the application for free using the register option. After login, the user is redirected to the upload page

3. **Upload**

   The user either drags a file to upload or selects one from his machine. The user then inputs the related file tags and
a passphrase to encrypt the file. A prompt is sent to the user when the file is uploaded.

Figure 11 – Upload Page

4. Retrieve File

There are 2 steps involved to retrieve a file. First, the user searches for the file. The user has two options to search for the file - search using the tags specified or using the name of the file. The file name option works only if the user is the original owner of the file.
After the file is found, the user inputs the passphrase used during the encryption to decrypt the file. The decrypted file is automatically downloaded on the user’s machine.
i. Node Modules –

This part discusses the various Node packages used to develop the project, their uses and justification.

1. Express

Express is the most popular NodeJS module. It provides a robust set of features to aid the development process. It automatically sets up the middleware to respond to HTTP requests, it allows segregation of different routes to perform different actions based on the HTTP request and URL and allows dynamic rendering of HTML pages by allowing passing of arguments to the front-end templates. **Justification** – Express prevents the need to setup the project from scratch and implements a lot of the basic functionalities needed by the project.

2. Passport

Passport is an authentication middleware developed for NodeJS. It can be used in any Express application without any modification to support login. It has multiple
authentication-based strategies such as using username and password, Facebook, Twitter, OAuth etc.  

**Justification** – Passport is extremely flexible and easy to use. It has a very well-developed documentation providing detailed information and examples on how to integrate it into a NodeJS project. Passport.js has a much cleaner and easier to use than any of the alternative authentication modules.

3. *Crypto-JS*

Crypto-JS is a growing collection of JavaScript implementation of standard Cryptographic techniques using best practices and techniques. It is fast, consistent and simple. It is used to encrypt and decrypt the contents of the documents using AES cipher algorithm. It is also used to hash the keywords obtained by the user using SHA-1 hashing algorithm.

**Justification** – Crypto-JS uses industry standard algorithms and allows us to manipulate their uses. It is extremely fast due to the way it has been implemented. It allows us to skip the implementation of the cryptographic algorithms and use them through this module instead.
4. **IPFS**

The IPFS module is a set of APIs used to setup and connect with the IPFS network to use its features including file upload, download, sharding etc.

**Justification** – It contains the complete set of functions needed to work with the IPFS network with proper documentation.

5. **Web3**

Web3 is a collection of JavaScript libraries which allows the user to interact with a local or remote Ethereum blockchain using different methods. It is used to execute a smart contract on the Ethereum blockchain.

**Justification** – Web3 is fast and allows the use of chrome extensions to connect to the digital wallet and the Ethereum Blockchain in a safe and secure manner.
3. Searchable Encryption

Searchable encryption is a technique in cryptography that enables searching for information in an encrypted document present in cloud storage which satisfying a prerequisite without decrypting the document to search. This involves three participants:

- **Data Owner**: The data owner is the original owner of information which is to be stored. The data owner is responsible for encrypting the data using an encryption key before uploading it to the cloud server.

- **Storage Provider**: This entity provides cloud storage services to its clients. It sets up a cloud server which stores the encrypted data given by the data owner. It is also responsible for managing and searching for the data for a given search query by a user. The encrypted data which fulfills the search query is returned back to the user thus, it is also responsible for data retrieval. The provider must not be aware of the type of information it stores during its upload or retrieval.
• *Data User*: This entity uses the service provided by the storage provider to retrieve specific encrypted data through a search query. The data user and data owner may be the same, however, the key used by data owner is needed to decrypt the retrieved information.

The data owner has *n* documents labeled as $D_1, D_2, ..., D_n$ that are to be stored on the cloud. The data owner encrypts these documents using his encryption key $k$ into $C=(C_1, C_2, ..., C_n)$ along with an Index $I$ for the documents. The data owner sends $(C,I)$ to the server. The Data User, to search for a keyword $w$, needs to generate a search token $t_w$ and send it to the server as a search query. The server uses $C, I$ and $t_w$ to search for the encrypted text $C_x$ containing the keywords given in the search token and returns it to the user. The data user then decrypts the text locally using the key $k$ obtained from the data owner [12].

### 3.1. Security in Searchable Encryption

Searchable encryption is secure if the following holds true:

1. The server is unable to gain any information about the document $D_i$ from its ciphertext $C_i$ for any $i$. This is possible as the document is encrypted using AES-256 for
the project with only the Data owner having the encryption key.

2. While searching, the server cannot learn anything about the decrypted documents and the search plain keyword except the search results. This is ensured by using SHA-1 hashing to hash the search keywords before sending to the server for processing. [12]

3.2. Workflow

The workflow consists of six stages: key generation, document pre-processing, encryption, search token generation, search and decryption. The details for each stage are given below. [13]
1. **Key Generation:** It involves generating the encryption/decryption key using symmetric key generation. The same key is used by the data owner to encrypt the document and data user to decrypt. This is done using the passphrase provided by the user using Crypto-JS to generate a 256-bit key to encrypt the document.

2. **Document Pre-processing:** This stage involves all the functions to pre-processing the document. This is done to start the encryption process. Before encryption, the pre-processing is done to get the keywords present in each document and build an index. Let $D$ be the set of documents to be uploaded. This step involves taking each document in $D$, assigning it a label and then creating an index table with each row having a document and its keywords. For example, consider 5 documents in $D = D_1, D_2, D_3, D_4, D_5$. An index table is built as shown in Table 1.
Table 1 – Table for mapping all keywords in a document

<table>
<thead>
<tr>
<th>Document</th>
<th>Document ID</th>
<th>Keywords</th>
</tr>
</thead>
<tbody>
<tr>
<td>D(_1)</td>
<td>1</td>
<td>(W_1, W_2, W_3)</td>
</tr>
<tr>
<td>D(_2)</td>
<td>2</td>
<td>(W_2, W_3)</td>
</tr>
<tr>
<td>D(_3)</td>
<td>3</td>
<td>(W_1, W_3, W_4, W_5)</td>
</tr>
<tr>
<td>D(_4)</td>
<td>4</td>
<td>(W_6, W_7, W_8)</td>
</tr>
<tr>
<td>D(_5)</td>
<td>5</td>
<td>(W_1, W_5, W_8, W_9)</td>
</tr>
</tbody>
</table>

After creating an index table, an inverted index table is created, mapping each keyword to all the documents it occurs in. For the above example, an inverted index table is shown in Table 2.

Table 2 - Inverted index table mapping keyword to all documents

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Document ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>(W_1)</td>
<td>1,3,5</td>
</tr>
<tr>
<td>(W_2)</td>
<td>1,2</td>
</tr>
<tr>
<td>(W_3)</td>
<td>1,2,3</td>
</tr>
<tr>
<td>(W_4)</td>
<td>3</td>
</tr>
<tr>
<td>(W_5)</td>
<td>3,5</td>
</tr>
<tr>
<td>(W_6)</td>
<td>4</td>
</tr>
<tr>
<td>(W_7)</td>
<td>4</td>
</tr>
<tr>
<td>(W_8)</td>
<td>4,5</td>
</tr>
<tr>
<td>(W_9)</td>
<td>9</td>
</tr>
</tbody>
</table>
3. *Encryption:* This stage is used to encrypt the documents and the index table. The encryption is done using the key generated during key generation stage. For a key \( k \), the encrypted document is stored as \( ENC(D_i) \). Moreover, the keywords are hashed using a hashing algorithm and are stored as \( H(W_i) \). For the above example, the encrypted document list is shown in Table 3 and the hashed keyword list in Table 4.

*Table 3 – Encrypted Document with Document ID*

<table>
<thead>
<tr>
<th>Document</th>
<th>Document ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>ENC(D₁)</td>
<td>1</td>
</tr>
<tr>
<td>ENC(D₂)</td>
<td>2</td>
</tr>
<tr>
<td>ENC(D₃)</td>
<td>3</td>
</tr>
<tr>
<td>ENC(D₄)</td>
<td>4</td>
</tr>
<tr>
<td>ENC(D₅)</td>
<td>5</td>
</tr>
</tbody>
</table>
4. **Search Token Generation**: This involves searching for a keyword in the encrypted documents stored. It contains the functions to create a search token from the data user to the server. The user inputs all the keywords it wants to search. The keywords are hashed again and the search token is generated as \( t = (t_1, t_2, \ldots, t_n) \) for \( n \) keywords. Once computed, the token is sent to the server in a search query.

5. **Search**: The server uses the search token and the hashed keyword index table and searches for the encrypted...
keyword. When $H(W_i)$ matches any keyword sent in search token $t$, the document ID is returned. The document containing the most matches is returned to the data user.

6. Decryption: The data user now decrypts the document obtained by using the symmetric key generated in the key generation stage which is obtained from the data owner.

In the application, we directly compute the Encrypted Document with Document Id (Table 3) and Hashed Keyword with Document Id (Table 4). The document id is the unique hash returned after uploading the file to IPFS node.
4. Results

The application has been successfully developed. The application makes use of IPFS to upload and retrieve data successfully. It is also protected against man in the middle attacks as the entire document is encrypted before it is sent to the server. The program also satisfies all the security requirements of Searchable Encryption. The server has no means to decipher the content of any document or the keywords associated with it. There were multiple challenges but the team managed to successfully overcome them. The final application is suitable for long term secure file storage. It is a security-based application with enforced encryption.
5. Challenges

The team faced several challenges while developing the application which are documented below.

- **Lack of Resources**: The biggest challenge the team encountered was the lack of resources and poor documentation. Blockchain technology and IPFS is a relatively new technology with changes coming frequently. This has led to different and sometimes contrasting resources. Moreover, the documentation for the newer versions was incomplete.

- **Rapidly changing technologies**: IPFS still had bugs and glitches and doesn’t work as expected all the time. Understanding and eliminating these problems was another challenge.

- **Lack of Experience**: Due to new technologies being used in the project, the team had to familiarize itself before any development can take place.
6. Limitations

There are some limitations to the application developed by the team.

- **Upload file size**: Despite implementing file sharding, the file size is limited due to front end JQuery code which results in undesirable characters being added to the file in case of a file larger than 1 MB.

- **Retrieval speed**: The retrieval speed for IPFS depends on the node where the file is stored. In case the file is not accessible through the node, it takes additional time to retrieve the file. The time may be up to 3 minutes to retrieve the file.

- **IPFS Changes**: IPFS is a rapidly changing technology and thus the project needs to be updated to support the new features and changes of IPFS regularly.

- **File type limitation**: Currently only text files are supported for upload due to the way encryption takes place. Graphics in file or image files are not supported and give errors.
7. Conclusion

The project is completed as per schedule. All preliminary features work. Currently, there are some limitations of the application developed however, the team will work on these to make a viable product for use. Initially, the progress was slow due to the challenges faced by the team in the early stages of the development project. Once that was completed the team made rapid development.
8. References


