SEARCHABLE KEYWORDS OVER ENCRYPTED CLOUD DATA

Submitted as a project report for mini-project

For the partial fulfilment of the degree of

Bachelor of Technology

in

Information Technology

by

Vikash Gopalak 2022ITB086

Ameer Hamza Khan 2022ITB087

Anurag Kumar 2022ITB094

Under the supervision of

Prof. Binanda Sengupta

Assistant Professor

Department of Information Technology

IIEST, Shibpur

April, 2024

Department of Information Technology

Indian Institute of Engineering Sceince and Technology,Shibpur

P.O. Botanic Garden,Howrah 711103,WB,India

+91 33 26684561-3 (3 lines)/0521-5(5 lines) X260

https://www.iiests.ac.in/IIEST/AcaUnitDetails/IT

IIEST, Shibpur

आई आई ई एस टि, शिवपुर

Erstwhile B E College (Estd 1856)

Date:

TO WHOM IT MAY CONCERN

This is to certify that Vikash Gopalak **2022ITB086,** Ameer Hamza Khan **2022ITB087** and Anurag Kumar **2022ITB094** have done their mini project on **Searchable Keywords over Encrypted Cloud Data.**

for the partial fulfilment of the degree of B.Tech. in Information Technology.

During this period they have completed the project. The report has fulfilled all the requirements as per the regulations of the institute and has reached the standard needed for submission.

_____

Prof. Binanda Sengupta
Assistant Professor,

Information Technology

_____

Prof. Prasun Ghoshal

HOD, Information Technology

Acknowledgement

We would like to express our profound gratitude to Prof. Binanda Sengupta, Assistant Professor, Information Technology, for his contributions to the completion of our project titled:

Searchable Keywords over Encrypted Cloud Data.

We want to express our thanks to Prof. Prasun Ghosal, HOD, Information Technology for keeping the mini project in this semester. In this aspect, we are grateful to the whole department and administration.

We would also like to mention all the batchmates for their valuable suggestions and the help they provided for the successful completion of the project.

We would like to acknowledge that this project was completed entirely by us and not by someone else.

Signature of the students: Vikash Gopalak _____

Ameer Hamza Khan _____

Anurag Kumar _____

Index

Chapter 1

Introduction

Searchable keyword over encrypted cloud data is a technique that allows users to search for specific keywords within their data stored in the cloud while keeping the data encrypted and secure from unauthorized access.

Naive Searching Technique

A conventional technique is here referred to as searching for the desired files among all the files stored on the server. This involves the decryption of all the files and then looking for the desired files among them.

Problems of using the Naive Searching Technique.

Time-consuming: File decryption and looking for the desired files increases the time complexity, thus making the search slow.

Storage-consuming: Clients will have numerous copies of the same file on their respective devices.

Vague Client-Server: There is no need for a client-server connection as all the files need to be retrieved from the server whenever the client looks for any file.

Ways to achieve Searchable Keywords over Encrypted Cloud Data.

Encrypted Inverted Index

Manages the creation of an inverted index from files and provides search functionality. It maps the encrypted keywords to their respective file identifiers, thus making the searching technique viable.

Files Encryption

Files are encrypted and then stored on the cloud server.

Socket Programming

The client and server are connected over an IP address and a common port through which the files are exchanged.

Chapter 2

Literature Survey

The three pillars of this project are:

1. AES-CBC: The Encryption Technique

2. Inverted Index: The Search Simplifier

3. Socket Programming: The Server-Client Sync

1. AES-CBC (Advanced Encryption Standard in Cipher Block Chaining mode)

It is a block cipher mode of operation that provides confidentiality by encrypting data in fixed-size blocks.

Encryption in AES-CBC mode

Initialization Vector (IV): Before encryption begins, an Initialization Vector is generated. The IV is a random value that is used to initialize the encryption process. It ensures that identical plaintext blocks will not encrypt to the same ciphertext blocks.

Dividing into Blocks: The plaintext message is divided into fixed-size blocks (typically 128 bits or 16 bytes) if it's not already in that form.
XOR with IV or Previous Cipher Block: Each plaintext block is XORed with the previous ciphertext block or the IV (for the first block).

Encryption: The result of the XOR operation is then encrypted using the AES algorithm with a chosen key. AES typically uses key sizes of 128, 192, or 256 bits.

Output: The encrypted ciphertext block becomes the input for the next block's XOR operation.

Finalization: Once all plaintext blocks have been processed, the final ciphertext is produced.

Decryption in AES-CBC mode is essentially the reverse process:

Initialization Vector (IV): The same IV used for encryption is required for decryption.
Decryption: Each ciphertext block is decrypted using the AES algorithm with the encryption key.

XOR with IV or Previous Cipher Block: The decrypted ciphertext block is XORed with the previous ciphertext block or the IV (for the first block).

Output: The result of the XOR operation yields the original plaintext block.

Finalization: Once all ciphertext blocks have been processed, the final decrypted plaintext is produced.

2. Inverted Index

An inverted index is a data structure used in information retrieval systems, such as search engines, to efficiently store and retrieve information about the occurrences of words (terms) within a collection of documents.

Document Collection: Initially, a collection of documents is indexed. These documents can be web pages, articles, books, or any other text-based content.

Tokenization: Each document is tokenized, meaning it is broken down into individual words or terms. Punctuation, whitespace, and other delimiters are typically used to separate terms.

Indexing: For each term in the document collection, an inverted index is created. This index maps each term to the documents that contain it. For example, the term "apple" might be associated with documents 1, 3, and 5.

Storage: Inverted indexes are typically stored in memory or on disk for fast retrieval. They can be implemented using various data structures, such as hash tables, balanced trees, or compressed bitmaps, depending on the specific requirements and characteristics of the application.

Query Processing: When a user enters a query, such as "apple pie recipe", the search engine looks up each term in the inverted index to find the documents that contain those terms. It then combines the results to return the most relevant documents to the user.

3. Socket Programming

Socket programming is a technique used for communication between processes, often across a network, using sockets, which are software endpoints that establish a connection between two nodes.

Creating Sockets: The process begins with creating sockets, which can be done using programming languages like Python, Java, C/C++, and others. Sockets can be of different types, such as TCP sockets for reliable, connection-oriented communication, or UDP sockets for unreliable, connectionless communication.

Binding and Listening: For server applications, a socket is bound to a specific port on the host machine. This allows the socket to listen for incoming connections. Once bound, the socket enters a listening state, waiting for clients to connect.

Connecting: For client applications, a socket is created and connected to the server's address and port. This establishes a connection between the client and server sockets, enabling communication.

Sending and Receiving Data: Once the connection is established, data can be sent and received between the client and server using the socket's send and receive functions. For TCP sockets, data transmission is reliable and ordered, while for UDP sockets, data is sent as individual packets without guaranteed delivery or ordering.

Handling Connections: Servers typically handle multiple client connections simultaneously by creating a new thread or process for each client, allowing for concurrent communication.

Chapter 3

Implementation of Searchable Keywords over Encrypted Cloud Data

1. Encryption and Decryption

Purpose: These functions provide basic encryption and decryption capabilities using a simple XOR-based technique.

Implementation:

XOR-based encryption takes each character of the message and XORs it with the corresponding character from the key.

Base64 encoding and decoding functions are used to encode binary data into ASCII characters and vice versa.

Files are encrypted using the Advanced Encryption Standard in Cipher Block Chaining(AES-CBC 128).

```
int AES_encrypt(string& file_name) {

    // Read plaintext from a file
    ifstream inputFile(file_name+".txt");
    if (!inputFile.is_open()) {
        cerr << "Error: Unable to open "<<file_name<<" file." << endl;
        return 1;
    }
    /*
    1. (istreambuf_iterator<char>(inputFile)): This part creates an istreambuf_iterator object,
    which is an iterator used to read characters from an input stream buffer (inputFile in this case).
    It reads characters of type char from the input stream.

    2. (istreambuf_iterator<char>()): This part creates another istreambuf_iterator object,
    but without any arguments. This creates an end-of-stream iterator, indicating the end of the input.
    */
    string plaintext((istreambuf_iterator<char>(inputFile)), (istreambuf_iterator<char>()));
    // convert string from  range between the two iterators.
    inputFile.close();


    // Encrypt the plaintext using AES CBC mode
    string ciphertext = encryptData(plaintext, key, iv);

    // Write ciphertext to an output file
    ofstream outputFile(file_name+".dat", ios::binary);
    if (!outputFile.is_open()) {
        cerr << "Error: Unable to create output file." << endl;
        return 1;
    }
    // we stored the initialization vector in file only then append the actual file
    outputFile.write(reinterpret_cast<const char*>(iv), sizeof(iv));
    outputFile.write(ciphertext.c_str(), ciphertext.length());
    outputFile.close();



    cout << "Encryption of "<<file_name<<" completed successfully.\n" << endl;
    return 0;
}
```

2. Inverted Index Class:

Purpose: Manages the creation of an inverted index from files and provides search functionality.

Data Structure: Uses an unordered map to store words as keys and sets of file names as values.

Methods:

create_inv_index: Creates the inverted index from files, tokenizing content and indexing words.

store_encrypted_invindex: Serializes and stores the encrypted inverted index to a file.

search: Searches for keywords in the encrypted index and retrieves associated file names.

```
class InvertedIndex {
private:
    unordered_map<string, unordered_set<string>> invIndex;
    string password;
public:
    vector<string> file_names;
    void create_inv_index() {
        int noFile;
        cout << "ENTER NUMBER OF FILES FOR OUTSOURCING: ";
        cin >> noFile;

        for (int i = 0; i < noFile; i++) {
            string file_name;
            cout << "ENTER FILE NAME: ";
            cin >> file_name;

            ifstream file(file_name+".txt");
            if (!file.is_open()) {
                cout << "Error opening file: " << file_name << endl;
                exit(1); // Exit with error code 1
            }

            // Read file contents into stringstream
            stringstream buffer;
            buffer << file.rdbuf();

            // Tokenize and index the file content
            makeIndex(buffer.str(), file_name);

            file_names.push_back(file_name+".dat");

            // encrypt the file using AES cbc
            AES_encrypt(file_name);

            file.close();
        }

        file_names.push_back("encrypted_inverted_index.txt");

    }

    // Function to tokenize and index the file content
    void makeIndex(const string& s, string Id) {
        string st = "";
        for (char c : s) {
            if ((c>='a' && c<='z') || (c>='A' && c<='Z') || (c>='0' && c<='9')) {
                if(c>='A' && c<='Z')
                c=tolower(c);
                st += c; // Convert characters to lowercase
            } else if (!st.empty()) {
                invIndex[st].insert(Id); // Store document ID for the current word
                st = ""; // Reset the word
            }
        }
        if (!st.empty()) {
            invIndex[st].insert(Id); // Store document ID for the last word
        }
    }
```

3. Main Function:

Purpose: Serves as the entry point of the program and orchestrates various operations.

User Interaction*: Prompts the user for a password and presents options for different operations.

Operation Handling:

Option 1: Encrypt files with the inverted index.

Option 2: Performs query retrieval by encrypting keywords and sending them to the server.

Option 3: Upload files to the server.

Option 4: Decrypts received files.

C++ code for Socket Programming

```
class server_connection {
private:
    int serverPort;
    string serverAddress;
public:
    server_connection(const string& address, int port) : serverAddress(address), serverPort(port) {}

    bool sendFiles(const vector<string>& fileNames) {
        // Create socket
        int clientSocket = socket(AF_INET, SOCK_STREAM, 0);
        if (clientSocket == -1) {
            cerr << "Error creating socket." << endl;
            return false;
        }

        // Server address
        struct sockaddr_in serverAddr;
        serverAddr.sin_family = AF_INET;
        serverAddr.sin_port = htons(serverPort);
        inet_pton(AF_INET, serverAddress.c_str(), &serverAddr.sin_addr);

        // Connect to server
        if (connect(clientSocket, (struct sockaddr*)&serverAddr, sizeof(serverAddr)) == -1) {
            cerr << "Error connecting to server." << endl;
            close(clientSocket);
            return false;
        }

        // Send files
        for (const string& fileName : fileNames) {
            ifstream file(fileName, ios::binary);
            if (!file.is_open()) {
                cerr << "Error opening file: " << fileName << endl;
                continue;
            }

            // Send file name length and file name
            uint32_t nameLen = fileName.length();
            nameLen = htonl(nameLen);
            send(clientSocket, &nameLen, sizeof(nameLen), 0);
            send(clientSocket, fileName.c_str(), fileName.length(), 0);

            // Send file contents
            stringstream buffer;
            buffer << file.rdbuf();
            string fileContents = buffer.str();
            uint32_t fileLen = fileContents.length();
            fileLen = htonl(fileLen);
            send(clientSocket, &fileLen, sizeof(fileLen), 0);
            send(clientSocket, fileContents.c_str(), fileContents.length(), 0);

            file.close();
        }

        // Close socket
        close(clientSocket);
        return true;
    }
```

Chapter 4
Depiction of the proposed model

Step 1: Setting Password for user authentication

```
ENTER PASSWORD
ameerhamza@khan123
```

Step 2: The client enters the number of files to be uploaded

```
ENTER OPTION OF CLIENT
PRESS
 1 --> FOR ENCRYPTING FILES with INVERTED INDEX
 2 --> FOR QUERY RETRIEVAL
 3 --> FOR UPLOADING FILES
 4 -->DECRYPTING RECIEVED FILES
1
ENCRYPTING FILES..
KEY generated
ENTER NUMBER OF FILES FOR OUTSOURCING: 3
```

Step 3: Encrypting Files and inverted index in parallel.

```
ENTER OPTION OF CLIENT
PRESS
 1 --> FOR ENCRYPTING FILES with INVERTED INDEX
 2 --> FOR QUERY RETRIEVAL
 3 --> FOR UPLOADING FILES
 4 -->DECRYPTING RECIEVED FILES
1
ENCRYPTING FILES..
KEY generated
ENTER NUMBER OF FILES FOR OUTSOURCING: 3
ENTER FILE NAME: file1
Encryption of file1 completed successfully.

ENTER FILE NAME: file2
Encryption of file2 completed successfully.

ENTER FILE NAME: file3
Encryption of file3 completed successfully.
```

Step 3: Uploading Encrypted Files to the Server.

```
3
UPLOADING FILES...
Files sent successfully.
```

Step 4: File received on the server side

```
ENTER OPTION OF SERVER
PRESS
 1 --> FOR RECEIVING DATA
 2 --> FOR QUERY SEARCH
 3--> FOR LOADING OF INVERTED INDEX
1
RECEIVE FILES...
Server listening on port 12345
Client connected: 127.0.0.1
Received file: file1.dat
Received file: file2.dat
Received file: file3.dat
Received file: encrypted_inverted_index.txt
Client disconnected.
```

Step 5: Loading encrypted inverted index on the server side

```
ENTER OPTION OF SERVER
PRESS
 1 --> FOR RECEIVING DATA
 2 --> FOR QUERY SEARCH
 3--> FOR LOADING OF INVERTED INDEX
3
LOADING INDEX...
AAAAAAA=AAAAAAA= --> file1
BwQJAEA=BwQJAEA= --> file2
BwQJAEM=BwQJAEM= --> file1
FwQOBAE=FwQOBAE= --> file2
FQUMFg==FQUMFg== --> file1 file2 file3
AAMQFxMPAAMQFxMP --> file3
CB4=CB4= --> file1 file2 file3
BwQJAEE=BwQJAEE= --> file3
INDEX LOADED
2
RECEIVING QUERY...
Server listening for queries on port 12345
Client connected: 127.0.0.1
```

Step 6: asking for the query to the server

```
2
QUERY RETRIEVAL
ENTER KEYWORDS TO RETRIEVE DATA (press (end) to terminate query at last )
vikas ameer anurag end
FwQOBAE=FwQOBAE= as query
AAAAAAA=AAAAAAA= as query
AAMQFxMPAAMQFxMP as query
Received file: file1
Received file: file3
Received file: file2
```

Step 7: Server receiving query

```
2
RECEIVING QUERY...
Server listening for queries on port 12345
Client connected: 127.0.0.1
```

Chapter 5

Conclusion and Future Scope

In conclusion, searchable keyword over encrypted cloud data combines privacy and optimised searchability. Utilizing encryption schemes, it allows users to securely search for specific keywords within their encrypted cloud-stored data. This addresses the growing need for data privacy in various domains, providing a secure solution for accessing sensitive information stored in the cloud.

Looking towards the future, there are several areas where the system can be further developed and improved.

These include:

1. **Multi-client system: Our project is based on one client-one server. In future, it can be scaled for multiple users on a single server.**

2. **Dynamic Insertion/Deletion: Our project currently works on a single-time outsourcing of files so we have to introduce a dynamic insertion/deletion of files from a user.**

3. **Encryption enhancement: we can improve our encryption for files and inverted index.**

4. **Privacy Concerns: We can improve the randomisation of encryption to make it more secure.**

References

1.Manoj Prabhakaran, Carl A.Gunter. Dynamic Searchable Encryption via Blind Storage.Accepted to IEEE Symposium on Security and Privacy(Oakland) 2014.

2. Song, D. X., Wagner, D., & Perrig, A. (2000). Practical techniques for searches on encrypted data. In IEEE Symposium on Security and Privacy, 2000. S&P 2000. Proceedings (pp. 44-55). IEEE.

3. Yu, S., Wang, C., Ren, K., & Lou, W. (2010). Achieving secure, scalable, and fine-grained data access control in cloud computing. In INFOCOM, 2010 Proceedings IEEE (pp. 1-9). IEEE.

Appendix

C++ codes:

1. Server main

```
#include <iostream>
#include <fstream>
#include <unordered_set>
#include <unordered_map>
#include <cstring>
#include <vector>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <unistd.h>
#include <sstream>
using namespace std;

unordered_map<string, unordered_set<string>> invIndex;
class FileReceiver {
private:
    int serverSocket;
    int port;
public:
    FileReceiver(int port) : port(port) {}

    bool receiveFiles() {
         // Create socket
        int serverSocket = socket(AF_INET, SOCK_STREAM, 0);
        if (serverSocket == -1) {
            cerr << "Error creating socket." << endl;
            exit(0);
        }


        // Server address
        struct sockaddr_in serverAddr;
        serverAddr.sin_family = AF_INET;
        serverAddr.sin_port = htons(port);
        serverAddr.sin_addr.s_addr = INADDR_ANY;

        // Bind socket
        if (bind(serverSocket, (struct sockaddr*)&serverAddr,
sizeof(serverAddr)) == -1) {
            cerr << "Error binding socket." << endl;
            close(serverSocket);
            exit(0);
        }

        // Listen
        if (listen(serverSocket, 5) == -1) {
            cerr << "Error listening on socket." << endl;
```

```
            close(serverSocket);
            exit(0);
        }

        cout << "Server listening on port " << port << endl;

        while (true) {
            // Accept incoming connection
            struct sockaddr_in clientAddr;
            socklen_t clientAddrLen = sizeof(clientAddr);
            int clientSocket = accept(serverSocket, (struct
sockaddr*)&clientAddr, &clientAddrLen);
            if (clientSocket == -1) {
                cerr << "Error accepting connection." << endl;
                close(serverSocket);
                return false;
            }

            cout << "Client connected: " <<
inet_ntoa(clientAddr.sin_addr) << endl;

            // Receive files
            while (true) {
                // Receive file name length
                uint32_t nameLen;
                int bytesReceived = recv(clientSocket, &nameLen,
sizeof(nameLen), 0);
                if (bytesReceived <= 0) {
                    cout << "Client disconnected." << endl;
                    break;
                }
                nameLen = ntohl(nameLen);

                // Receive file name
                char fileName[nameLen + 1];
                bytesReceived = recv(clientSocket, fileName, nameLen,
0);
                if (bytesReceived <= 0) {
                    cout << "Client disconnected." << endl;
                    break;
                }
                fileName[nameLen] = '\0';

                // Receive file content length
                uint32_t fileLen;
                bytesReceived = recv(clientSocket, &fileLen,
sizeof(fileLen), 0);
                if (bytesReceived <= 0) {
                    cout << "Client disconnected." << endl;
                    break;
                }
                fileLen = ntohl(fileLen);

                // Receive file content
                char fileContents[fileLen + 1];
                bytesReceived = recv(clientSocket, fileContents,
fileLen, 0);
                if (bytesReceived <= 0) {
```

```
                    cout << "Client disconnected." << endl;
                    break;
                }
                fileContents[fileLen] = '\0';

                // Write received file content to file
                ofstream outFile(fileName, ios::binary);
                outFile.write(fileContents, fileLen);
                outFile.close();

                cout << "Received file: " << fileName << endl;
            }

            // Close client socket
            close(clientSocket);
            break;
        }

        // Close server socket
        close(serverSocket);
        return true;
    }

void receive_file(){
    // Start receiving files
    bool success = receiveFiles();
    if (!success) {
        cerr << "Error receiving files." << endl;
        exit(0);
    }
}


void receive_query(){

     // Create socket
        int serverSocket = socket(AF_INET, SOCK_STREAM, 0);
        if (serverSocket == -1) {
            cerr << "Error creating socket." << endl;
            exit(0);
        }

        // Server address
        struct sockaddr_in serverAddr;
        serverAddr.sin_family = AF_INET;
        serverAddr.sin_port = htons(port);
        serverAddr.sin_addr.s_addr = INADDR_ANY;

        // Bind socket
        if (bind(serverSocket, (struct sockaddr*)&serverAddr,
sizeof(serverAddr)) == -1) {
            cerr << "Error binding socket." << endl;
            close(serverSocket);
            exit(0);
        }
        // Listen
        if (listen(serverSocket, 5) == -1) {
            cerr << "Error listening on socket." << endl;
```

```
            close(serverSocket);
            exit(0);
        }

    cout << "Server listening for queries on port " << port << endl;


    while (true) {
        // Accept incoming connection
        struct sockaddr_in clientAddr;
        socklen_t clientAddrLen = sizeof(clientAddr);
        int clientSocket = accept(serverSocket, (struct
sockaddr*)&clientAddr, &clientAddrLen);
        if (clientSocket == -1) {
            cerr << "Error accepting connection." << endl;
            close(serverSocket);
            return;
        }

        cout << "Client connected: " << inet_ntoa(clientAddr.sin_addr)
<< endl;


        // Receive the size of query
        uint32_t querysize;
        int bytesReceived = recv(clientSocket, &querysize,
sizeof(querysize), 0);
        if (bytesReceived <= 0) {
            cout << "Client disconnected." << endl;
            break;
        }
        querysize = ntohl(querysize);
        unordered_set<string> file_names;

       // Receive query's
            while (querysize--) {
            uint32_t queryLen;
            int bytesReceived = recv(clientSocket, &queryLen,
sizeof(queryLen), 0);
            if (bytesReceived <= 0) {
                cout << "Client disconnected." << endl;
                break;
            }
            queryLen = ntohl(queryLen);

            // Receive query
            char queryBuffer[queryLen + 1];
            bytesReceived = recv(clientSocket, queryBuffer, queryLen,
0);
            if (bytesReceived <= 0) {
                cout << "Client disconnected." << endl;
                break;
            }
            queryBuffer[queryLen] = '\0';

            string query(queryBuffer);

            // Get response for query
```

```
            string result;
            if(invIndex.find(query)!=invIndex.end()){
            for (const auto& response : invIndex[query]) {
                result = response;
                file_names.insert(result);
            }
            }

            }
            // code here
            send_files(clientSocket, file_names);

            // Close client socket
            close(clientSocket);
}
}
void send_files(int clientSocket, const  unordered_set<string>&
file_names) {
    uint32_t files_size = file_names.size();
    files_size = htonl(files_size);
    send(clientSocket, &files_size, sizeof(files_size), 0);

    for (const string& fileName : file_names) {
        ifstream file(fileName+".dat", ios::binary);
        if (!file.is_open()) {
            cerr << "Error opening file: " << fileName << endl;
            continue;
        }

        // Send file name length and file name
        uint32_t nameLen = fileName.length();
        nameLen = htonl(nameLen);
        send(clientSocket, &nameLen, sizeof(nameLen), 0);
        send(clientSocket, fileName.c_str(), fileName.length(), 0);

        // Send file contents
        stringstream buffer;
        buffer << file.rdbuf();
        string fileContents = buffer.str();
        uint32_t fileLen = fileContents.length();
        fileLen = htonl(fileLen);
        send(clientSocket, &fileLen, sizeof(fileLen), 0);
        send(clientSocket, fileContents.c_str(), fileContents.length(),
0);

        file.close();
    }
}
};




// Load inverted index
void load_inverted_index(const string& filename,unordered_map<string,
unordered_set<string>>& invIndex) {
    ifstream inFile(filename);
    if (!inFile.is_open()) {
```

```
        cerr << "Error opening file for deserialization: " << filename
<< endl;
        return;
    }
    string word;
    string docIdStr;
    while (inFile >> word) {
        // Read document IDs until the end of line
        getline(inFile, docIdStr);
        istringstream iss(docIdStr);
        while (iss >> docIdStr) {
            invIndex[word].insert(docIdStr);
        }
    }
    inFile.close();
}

    void printInvertedIndex() {
        for (const auto& word : invIndex) {
            cout << word.first << " --> ";
            for (const string& docId : word.second) {
                cout << docId << " ";
            }
            cout << endl;
        }
    }
int main() {

    // Port on which the server listens for incoming file transfers
    int serverPort = 12345; // Change to your desired port

    // Initialize file receiver
    FileReceiver fileReceiver(serverPort);
    int token;
    cout<<"ENTER OPTION OF SERVER \n";
    cout<<"PRESS \n 1 --> FOR RECEIVING DATA\n 2 --> FOR QUERY SEARCH\n
3--> FOR LOADING OF INVERTED INDEX\n";
    while(true){
    cin>>token;
    switch(token){
        case 1:
        cout<<"RECEIVE FILES... \n";
        fileReceiver.receive_file();
        break;
        case 2:
        cout<<"RECEIVING QUERY... \n";
        fileReceiver.receive_query();
        cout<<"RESPONSE SENT\n";
        break;
        case 3:
        cout<<"LOADING INDEX... \n";
        load_inverted_index("encrypted_inverted_index.txt", invIndex);
        printInvertedIndex();
        cout<<"INDEX LOADED\n";
        break;
        default :return 0;
    }
    }
```

```
    return 0;
}
```

2. Client_Socket.cpp

```
#include <iostream>
#include <fstream>
#include <sstream>
#include <vector>
#include <string>
#include <cstring>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <unistd.h>

using namespace std;
vector<string> received_files;

class server_connection {
private:
    int serverPort;
    string serverAddress;
public:
    server_connection(const string& address, int port) :
serverAddress(address), serverPort(port) {}

    bool sendFiles(const vector<string>& fileNames) {
        // Create socket
        int clientSocket = socket(AF_INET, SOCK_STREAM, 0);
        if (clientSocket == -1) {
            cerr << "Error creating socket." << endl;
            return false;
        }

        // Server address
        struct sockaddr_in serverAddr;
        serverAddr.sin_family = AF_INET;
        serverAddr.sin_port = htons(serverPort);
        inet_pton(AF_INET, serverAddress.c_str(), &serverAddr.sin_addr);

        // Connect to server
        if (connect(clientSocket, (struct sockaddr*)&serverAddr,
sizeof(serverAddr)) == -1) {
            cerr << "Error connecting to server." << endl;
            close(clientSocket);
            return false;
        }

        // Send files
        for (const string& fileName : fileNames) {
            ifstream file(fileName, ios::binary);
            if (!file.is_open()) {
                cerr << "Error opening file: " << fileName << endl;
                continue;
            }
```

```
            // Send file name length and file name
            uint32_t nameLen = fileName.length();
            nameLen = htonl(nameLen);
            send(clientSocket, &nameLen, sizeof(nameLen), 0);
            send(clientSocket, fileName.c_str(), fileName.length(), 0);

            // Send file contents
            stringstream buffer;
            buffer << file.rdbuf();
            string fileContents = buffer.str();
            uint32_t fileLen = fileContents.length();
            fileLen = htonl(fileLen);
            send(clientSocket, &fileLen, sizeof(fileLen), 0);
            send(clientSocket, fileContents.c_str(),
fileContents.length(), 0);

            file.close();
        }

        // Close socket
        close(clientSocket);
        return true;
    }

    bool send_query(const vector<string>& querys){
        // Create socket
        int clientSocket = socket(AF_INET, SOCK_STREAM, 0);
        if (clientSocket == -1) {
            cerr << "Error creating socket." << endl;
            return false;
        }

        // Server address
        struct sockaddr_in serverAddr;
        serverAddr.sin_family = AF_INET;
        serverAddr.sin_port = htons(serverPort);
        inet_pton(AF_INET, serverAddress.c_str(), &serverAddr.sin_addr);

        // Connect to server
        if (connect(clientSocket, (struct sockaddr*)&serverAddr,
sizeof(serverAddr)) == -1) {
            cerr << "Error connecting to server." << endl;
            close(clientSocket);
            return false;
        }

      // sending query size
       uint32_t queryLen = querys.size();
       queryLen = htonl(queryLen);
       send(clientSocket, &queryLen, sizeof(queryLen), 0);


    for (const string& query : querys) {
        // Send query length and query
        uint32_t queryLen = query.length();
        queryLen = htonl(queryLen);
        send(clientSocket, &queryLen, sizeof(queryLen), 0);
```

```
        send(clientSocket, query.c_str(), query.length(), 0);

        // // Receive response length
        // uint32_t responseLen;
        // recv(clientSocket, &responseLen, sizeof(responseLen), 0);
        // responseLen = ntohl(responseLen);

        // // Receive response
        // char responseBuffer[responseLen + 1];
        // recv(clientSocket, responseBuffer, responseLen, 0);
        // responseBuffer[responseLen] = '\0';

        // cout << "Query: " << query << " Answer: " << responseBuffer
<< endl;

        // code

    }
        receive_files(clientSocket);
        // Close socket
        close(clientSocket);
        return true;
    }

    void receive_files(int clientSocket) {
    uint32_t files_size;
    recv(clientSocket, &files_size, sizeof(files_size), 0);
    files_size = ntohl(files_size);

    while (files_size--) {
        // Receive file name length
        uint32_t nameLen;
        recv(clientSocket, &nameLen, sizeof(nameLen), 0);
        nameLen = ntohl(nameLen);

        // Receive file name
        char fileName[nameLen + 1];
        recv(clientSocket, fileName, nameLen, 0);
        fileName[nameLen] = '\0';

        received_files.push_back(string(fileName));
        // Receive file content length
        uint32_t fileLen;
        recv(clientSocket, &fileLen, sizeof(fileLen), 0);
        fileLen = ntohl(fileLen);

        // Receive file content
        char fileContents[fileLen + 1];
        recv(clientSocket, fileContents, fileLen, 0);
        fileContents[fileLen] = '\0';

        // Write received file content to file
        ofstream outFile(string(fileName)+".dat", ios::binary);
        outFile.write(fileContents, fileLen);
        outFile.close();

        cout << "Received file: " << fileName << endl;
    }
```

```
}

};

int sendFiles_to_server(vector<string>& content,bool flag) {
    // Server address and port
    string serverAddress = "127.0.0.1"; // Loopback address for local
communication
    int serverPort = 12345; // Change to your server port

    // Initialize file sender
    server_connection fileSender(serverAddress, serverPort);

    bool success =
flag?fileSender.sendFiles(content):fileSender.send_query(content);

    if (success) {
        cout << "Files sent successfully." << endl;
    } else {
        cerr << "Error sending files." << endl;
    }

    return 0;
}
```

3. Encrypt.cpp

```
#include<iostream>
#include<fstream>
#include<openssl/evp.h>
#include<openssl/rand.h>
#include<vector>

using namespace std;

unsigned char key[16], iv[16]; // as we use 128 cbc means 16 byte

// Function to encrypt data using AES CBC mode
string encryptData(const string& plaintext, const unsigned char* key,
const unsigned char* iv) {
    //creates a new cipher context for encryption
    EVP_CIPHER_CTX* ctx = EVP_CIPHER_CTX_new();
    /*
      initializes the encryption operation with the specified cipher
algorithm
      (EVP_aes_128_cbc() in this case), key, and initialization vector
(IV).
      CBC (Cipher Block Chaining) mode is used here.
    */
    EVP_EncryptInit_ex(ctx, EVP_aes_128_cbc(), nullptr, key, iv);
    /*
    Memory is allocated for the ciphertext buffer.
    The length of the ciphertext buffer is calculated as
plaintext.length() + EVP_MAX_BLOCK_LENGTH.
    This accounts for the maximum possible size of the ciphertext, which
might be larger than the plaintext due to padding.
```

```
    */
    int ciphertextLen = plaintext.length() + EVP_MAX_BLOCK_LENGTH;
    unsigned char* ciphertext = new unsigned char[ciphertextLen];

    /*
    performs the encryption operation.
    It encrypts the input plaintext (plaintext.c_str()) and writes the
encrypted data to the ciphertext buffer.
    The len parameter is used to store the actual length of the output.
    */

    int len;
    EVP_EncryptUpdate(ctx, ciphertext, &len, reinterpret_cast<const
unsigned char*>(plaintext.c_str()), plaintext.length());
    int ciphertextLen1 = len;

    /*
    completes the encryption operation.
    It writes any remaining ciphertext bytes to the ciphertext buffer.
    The len parameter is updated with the number of bytes written
    */
    EVP_EncryptFinal_ex(ctx, ciphertext + len, &len);
    // make ciphertext actual length
    ciphertextLen1 += len;


    // The ciphertext is converted to a string using its constructor,
    // which takes a pointer to the ciphertext buffer and its length
    string encryptedData(reinterpret_cast<char*>(ciphertext),
ciphertextLen1);

    // deallocating memory of ciphertext
    delete[] ciphertext;

    // The cipher context is freed using EVP_CIPHER_CTX_free()
    // to release any resources allocated during encryption
    EVP_CIPHER_CTX_free(ctx);

    return encryptedData;
}
void key_generate(){
    // Generate a random key and IV
    RAND_bytes(key, sizeof(key));
    RAND_bytes(iv, sizeof(iv));
    cout<< "KEY generated\n";
    // Write key to a text file
    ofstream keyFile("key.txt");
    if (!keyFile.is_open()) {
        cerr << "Error: Unable to create key file.\n" << endl;
        exit(0);
    }
    keyFile.write(reinterpret_cast<const char*>(key), sizeof(key));
    keyFile.close();
}
int AES_encrypt(string& file_name) {

    // Read plaintext from a file
    ifstream inputFile(file_name+".txt");
```

```
    if (!inputFile.is_open()) {
        cerr << "Error: Unable to open "<<file_name<<" file." << endl;
        return 1;
    }
    /*
    1. (istreambuf_iterator<char>(inputFile)): This part creates an
istreambuf_iterator object,
    which is an iterator used to read characters from an input stream
buffer (inputFile in this case).
    It reads characters of type char from the input stream.

    2. (istreambuf_iterator<char>()): This part creates another
istreambuf_iterator object,
    but without any arguments. This creates an end-of-stream iterator,
indicating the end of the input.
    */
    string plaintext((istreambuf_iterator<char>(inputFile)),
(istreambuf_iterator<char>()));
    // convert string from  range between the two iterators.
    inputFile.close();


    // Encrypt the plaintext using AES CBC mode
    string ciphertext = encryptData(plaintext, key, iv);

    // Write ciphertext to an output file
    ofstream outputFile(file_name+".dat", ios::binary);
    if (!outputFile.is_open()) {
        cerr << "Error: Unable to create output file." << endl;
        return 1;
    }
    // we stored the initialization vector in file only then append the
actual file
    outputFile.write(reinterpret_cast<const char*>(iv), sizeof(iv));
    outputFile.write(ciphertext.c_str(), ciphertext.length());
    outputFile.close();



    cout << "Encryption of "<<file_name<<" completed successfully.\n" <<
endl;
    return 0;
}
/*
g++ -o program encrypt.cpp -lssl -lcrypto
./program
*/
```

4.Client_main.cpp

```
#include <iostream>
#include <fstream>
#include <sstream>
#include <unordered_map>
#include <unordered_set>
#include <vector>
```

```
#include <string>
#include "encrypt.cpp"
#include "decrypt.cpp"
#include "client_socket.cpp"
using namespace std;


class personal_encyption{

// Simple XOR-based encryption function
std::string encrypt(const std::string& message, const std::string& key)
{
    std::string encrypted;
    for (size_t i = 0; i < message.size(); ++i) {
        encrypted += message[i] ^ key[i % key.size()]; // XOR with
corresponding character from key
    }
    return encrypted;
}

// Simple XOR-based decryption function
std::string decrypt(const std::string& encrypted, const std::string&
key) {
    std::string decrypted;
    for (size_t i = 0; i < encrypted.size(); ++i) {
        decrypted += encrypted[i] ^ key[i % key.size()]; // XOR with
corresponding character from key
    }
    return decrypted;
}


// Base64 encoding table
const std::string base64_chars =
    "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
    "abcdefghijklmnopqrstuvwxyz"
    "0123456789+/";

// Function to encode a string to Base64
std::string base64_encode(const std::string &input) {
    std::string encoded;
    size_t i = 0;
    size_t j = 0;
    unsigned char char_array_3[3];
    unsigned char char_array_4[4];

    for (auto c : input) {
        char_array_3[i++] = c;
        if (i == 3) {
            char_array_4[0] = (char_array_3[0] & 0xfc) >> 2;
            char_array_4[1] = ((char_array_3[0] & 0x03) << 4) +
((char_array_3[1] & 0xf0) >> 4);
            char_array_4[2] = ((char_array_3[1] & 0x0f) << 2) +
((char_array_3[2] & 0xc0) >> 6);
            char_array_4[3] = char_array_3[2] & 0x3f;

            for (i = 0; i < 4; i++) {
                encoded += base64_chars[char_array_4[i]];
```

```
            }
            i = 0;
        }
    }

    if (i) {
        for (j = i; j < 3; j++) {
            char_array_3[j] = '\0';
        }

        char_array_4[0] = (char_array_3[0] & 0xfc) >> 2;
        char_array_4[1] = ((char_array_3[0] & 0x03) << 4) +
((char_array_3[1] & 0xf0) >> 4);
        char_array_4[2] = ((char_array_3[1] & 0x0f) << 2) +
((char_array_3[2] & 0xc0) >> 6);

        for (j = 0; j < i + 1; j++) {
            encoded += base64_chars[char_array_4[j]];
        }

        while (i++ < 3) {
            encoded += '=';
        }
    }

    return encoded;
}

// Function to decode a Base64 string
std::string base64_decode(const std::string &encoded_string) {
    size_t in_len = encoded_string.size();
    size_t i = 0;
    size_t j = 0;
    int in_ = 0;
    unsigned char char_array_4[4], char_array_3[3];
    std::string decoded;

    while (in_len-- && (encoded_string[in_] != '=') &&
           (isalnum(encoded_string[in_]) || (encoded_string[in_] == '+')
|| (encoded_string[in_] == '/'))) {
        char_array_4[i++] = encoded_string[in_];
        in_++;
        if (i == 4) {
            for (i = 0; i < 4; i++) {
                char_array_4[i] = base64_chars.find(char_array_4[i]);
            }

            char_array_3[0] = (char_array_4[0] << 2) + ((char_array_4[1]
& 0x30) >> 4);
            char_array_3[1] = ((char_array_4[1] & 0xf) << 4) +
((char_array_4[2] & 0x3c) >> 2);
            char_array_3[2] = ((char_array_4[2] & 0x3) << 6) +
char_array_4[3];

            for (i = 0; i < 3; i++) {
                decoded += char_array_3[i];
            }
            i = 0;
```

```
        }
    }

    if (i) {
        for (j = i; j < 4; j++) {
            char_array_4[j] = 0;
        }

        for (j = 0; j < 4; j++) {
            char_array_4[j] = base64_chars.find(char_array_4[j]);
        }

        char_array_3[0] = (char_array_4[0] << 2) + ((char_array_4[1] &
0x30) >> 4);
        char_array_3[1] = ((char_array_4[1] & 0xf) << 4) +
((char_array_4[2] & 0x3c) >> 2);

        for (j = 0; j < i - 1; j++) {
            decoded += char_array_3[j];
        }
    }

    return decoded;
}
public:
std::string key;
  personal_encyption(string key)
  {
    this->key=key;
  }

  string encryption(string message){

    string a=base64_encode(encrypt(message, key));
    if(a.length()>=16)
    return a.substr(0,16);
    return a+a.substr(0,16-a.length());

  }

  string decryption(string cyphertext)
  {
    return decrypt(base64_decode(cyphertext), key);
  }
};


class InvertedIndex {
private:
    unordered_map<string, unordered_set<string>> invIndex;
    string password;
public:
    vector<string> file_names;
    void create_inv_index() {
        int noFile;
        cout << "ENTER NUMBER OF FILES FOR OUTSOURCING: ";
        cin >> noFile;
```

```
        for (int i = 0; i < noFile; i++) {
            string file_name;
            cout << "ENTER FILE NAME: ";
            cin >> file_name;

            ifstream file(file_name+".txt");
            if (!file.is_open()) {
                cout << "Error opening file: " << file_name << endl;
                exit(1); // Exit with error code 1
            }

            // Read file contents into stringstream
            stringstream buffer;
            buffer << file.rdbuf();

            // Tokenize and index the file content
            makeIndex(buffer.str(), file_name);

            file_names.push_back(file_name+".dat");

            // encrypt the file using AES cbc
            AES_encrypt(file_name);

            file.close();
        }

        file_names.push_back("encrypted_inverted_index.txt");

    }

    // Function to tokenize and index the file content
    void makeIndex(const string& s, string Id) {
        string st = "";
        for (char c : s) {
            if ((c>='a' && c<='z') || (c>='A' && c<='Z') || (c>='0' &&
c<='9')) {
                if(c>='A' && c<='Z')
                c=tolower(c);
                st += c; // Convert characters to lowercase
            } else if (!st.empty()) {
                invIndex[st].insert(Id); // Store document ID for the
current word
                st = ""; // Reset the word
            }
        }
        if (!st.empty()) {
            invIndex[st].insert(Id); // Store document ID for the last
word
        }
    }

    void printInvertedIndex() {
        for (const auto& word : invIndex) {
            cout << word.first << " --> ";
            for (const string& docId : word.second) {
                cout << docId << " ";
            }
            cout << endl;
```

```
        }
    }

    // Serialize the encrypted inverted index to a file
    void store_encrypted_invindex(const string& filename) {
        personal_encyption enc_dec(password);
        ofstream outFile(filename);
        if (!outFile.is_open()) {
            cerr << "Error opening file for serialization: " << filename
<< endl;
            return;
        }
        for (const auto& word : invIndex) {
            outFile << enc_dec.encryption(word.first) << " ";
            // Encrypt document IDs and store them in the file
            for (const string& file_id : word.second) {
                outFile << file_id <<" ";
            }
            outFile << endl;
        }
        outFile.close();
    }

    // Function to search for a keyword in the encrypted inverted index
    void search(const string& keyword) {
        // Encrypt keyword
        personal_encyption enc_dec(password);
        string encryptedKeyword = enc_dec.encryption(keyword);

        // Perform search using encrypted keyword
        // auto it = invIndex.find(encryptedKeyword);
        auto it = invIndex.find(keyword);

        if (it != invIndex.end()) {
            cout << "Keyword found. Files present in location: ";
            for (const string& loc : it->second) {
                cout << loc << " ";
            }
            cout << endl;
        } else {
            cout << "Keyword not found." << endl;
        }
    }

    void get_password(const string password){
        this->password=password;
    }
};

int main() {

    InvertedIndex indexing;
    string pass,keyword;
    vector<string> querys;
    cout<<"ENTER PASSWORD\n";
    cin>>pass;
    personal_encyption hash(pass);
```

```
    int token;
    cout<<"ENTER OPTION OF CLIENT \n";
    cout<<"PRESS \n 1 --> FOR ENCRYPTING FILES with INVERTED INDEX\n 2 -
-> FOR QUERY RETRIEVAL\n 3 --> FOR UPLOADING FILES\n 4 -->DECRYPTING
RECIEVED FILES\n";
    while(true){
    cin>>token;
    switch(token){
        case 1:
        cout<<"ENCRYPTING FILES.. \n";
        indexing.get_password(pass);
        key_generate();

       // Create index from files
        indexing.create_inv_index();


       // Print index
       // indexing.printInvertedIndex();

       // Serialize and store the encrypted index to a file

indexing.store_encrypted_invindex("encrypted_inverted_index.txt");

        break;
        case 2:
        cout<<"QUERY RETRIEVAL \n";
        // Search for a keyword
         cout<<"ENTER KEYWORDS TO RETRIEVE DATA (press (end) to
terminate query at last )\n";
         cin>>keyword;
        while(keyword!="end"){
        string enc_key=hash.encryption(keyword);
        cout<<enc_key<<" as query \n";
        querys.push_back(enc_key);
        cin>>keyword;
    }

    sendFiles_to_server(querys,false);
    cout<< "query received\n";
        break;

        case 3:
        cout<<"UPLOADING FILES... \n";
        // upload files to server
        sendFiles_to_server(indexing.file_names,true);
        break;
        case 4:
        cout<<"DECRYPTING RECIEVED FILES\n";
        AES_decrypt(received_files);
        break;
        default :return 0;
    }
    }

    return 0;
}
/*
```

```
ameer123@khan
3
file1.txt
file2.txt
file3.txt

g++ -o program client_main.cpp -lssl -lcrypto
./program

cd /mnt/c/users/adnan/C_Tutorials/project/searchable_encryption
*/
```

5.decrypt.cpp

```
#include<iostream>
#include<fstream>
#include<openssl/evp.h>
#include<vector>
using namespace std;

// Function to decrypt data using AES CBC mode
string decryptData(const string& ciphertext, const unsigned char* key,
const unsigned char* iv) {
    EVP_CIPHER_CTX* ctx = EVP_CIPHER_CTX_new();
    EVP_DecryptInit_ex(ctx, EVP_aes_128_cbc(), nullptr, key, iv);

    int plaintextLen = ciphertext.length() + EVP_MAX_BLOCK_LENGTH; //
Make room for padding
    unsigned char* plaintext = new unsigned char[plaintextLen];

    int len;
    EVP_DecryptUpdate(ctx, plaintext, &len, reinterpret_cast<const
unsigned char*>(ciphertext.c_str()), ciphertext.length());
    int plaintextLen1 = len;

    // Finalize decryption
    int finalLen;
    EVP_DecryptFinal_ex(ctx, plaintext + len, &finalLen);
    plaintextLen1 += finalLen;

    string decryptedData(reinterpret_cast<char*>(plaintext),
plaintextLen1);

    delete[] plaintext;
    EVP_CIPHER_CTX_free(ctx);

    return decryptedData;
}

int AES_decrypt(vector<string>& encrypted_files) {

    // Read key from the key file
    ifstream keyFile("key.txt", ios::binary);
    if (!keyFile.is_open()) {
        cerr << "Error: Unable to open key file." << endl;
        return 1;
```

```
    }
    unsigned char key[16];
    keyFile.read(reinterpret_cast<char*>(key), sizeof(key));
    keyFile.close();


    for(string file:encrypted_files){
    // Read IV and ciphertext from the encrypted file
    ifstream inputFile(file+".dat", ios::binary);
    if (!inputFile.is_open()) {
        cerr << "Error: Unable to open encrypted file." << endl;
        return 1;
    }

    unsigned char iv[16];
    inputFile.read(reinterpret_cast<char*>(iv), sizeof(iv));
    string ciphertext((istreambuf_iterator<char>(inputFile)),
(istreambuf_iterator<char>()));
    inputFile.close();

    // Decrypt the ciphertext using AES CBC mode
    string decryptedText = decryptData(ciphertext, key, iv);
    // Write the decrypted plaintext to a file

    ofstream outputFile(file+".txt");
    if (!outputFile.is_open()) {
        cerr << "Error: Unable to create output file." << endl;
        return 1;
    }
    outputFile << decryptedText;
    outputFile.close();
    }
    cout << "Decryption of files completed successfully.\n" << endl;
    return 0;
}
/*
g++ -o program decrypt.cpp -lssl -lcrypto
./program

g++ -o program client_main.cpp -lssl -lcrypto
*/
```