**Achieving Secure, Universal, & Fine-Grained Query Results Verification for Secure Search Scheme over Encrypted Cloud Data**

**1. INTRODUCTION**

Distributed processing is a model for allow universal, helpful, on-request organize access to a mutual pool of configurable registering possessions (e.g., systems, servers, stockpiling, applications, & administrations) that can be quickly provisioned & discharged with negligible administration exertion or specialist organization collaboration. Driven by the bounteous advantages brought by the distributed computing, for example, cost sparing, fast organization, adaptable asset setup, & so on., an ever improving no. of ventures & individual clients are considering moving their private information & local applications to the cloud server. A matter of open concern is the manner by which to ensure the security of information that is outsourced to a remote cloud server & splits far from the immediate control of information proprietors. Encryption on private information before outsourcing is a compelling measure to ensure information secrecy. Be that as it may, encoded information make compelling information recovery an exceptionally difficult assignment; to address the test (i.e., look on encoded information), Song et al. to start with presented the idea of accessible encryption & proposed a down to earth strategy that enables clients to seek over scrambled information through encoded question catchphrases. Afterward, numerous accessible encryption plans were proposed in light of symmetric key & open key setting to fortify security & enhance question proficiency. As of late, with the developing notoriety of distributed computing, how to safely & effectively seek over encoded cloud information turns into an examination center. Some methodologies have been proposed in light of customary accessible encryption plans, which expect to ensure information security & question protective measures with better inquiry effective for distributed computing. In any case, these plans depend on a perfect presumption that the cloud server is a "legitimate however inquisitive" element & keeps hearty & secure programming/equipment conditions. Therefore, right & finish question comes about dependably are unexceptionally come back from the cloud server when an inquiry closes without fail. Be that as it may, in down to earth applications, the cloud server may return wrong or fragmented question comes about once he acts insincerely for illicit benefits, for example, sparing calculation & correspondence cost or because of conceivable programming/equipment disappointment of the server. Subsequently, the above actuality ordinarily propels information clients to check the accuracy & fulfillment of question comes about. A few analysts proposed to incorporate the question comes about confirmation components to their safe inquiry plans, (e.g., implanting check data into the predetermined secure records or question comes about). After accepting inquiry comes about, information clients utilize determined confirmation data to check their correctess & culmination. There are two constraints in these plans: 1) These confirmation instruments give a coarsegrained check, i.e., if the question result set contains all qualified & right information records, at that point these plans answer yes, generally answer no. Consequently, if the check calculation yields no, an information client needs to prematurely end the unscrambling for all question comes about regardless of just a single inquiry result is off base. 2) These check components are for the most part firmly coupled to comparing secure inquiry developments & have not all inclusiveness. In an inquiry procedure, for a returned question comes about set that contains various encoded information documents, an information client may wish to confirm the rightness of each scrambled information record (along these lines, he can evacuate erroneous outcomes & hold the right ones as the ultima question results) or needs to check what number of or which qualified information records are not returned on earth if the cloud server purposefully overlooks some question comes about. These data can be viewed as a hard confirmation to rebuff the cloud server. This is trying to accomplish the fine-grained confirmations since the question & check are authorized in the scrambled condition. We proposed a safe & fine-grained question comes about check conspire by building the confirmation protest for encoded outsourced information documents. At the point when a question closes, the inquiry comes about set alongside the relating check protest are returned together, by which the inquiry client can precisely confirm: 1) the rightness of each scrambled information document in the outcomes set; 2) what number of qualified information records are not returned & 3) which qualified information documents are not returned. Besides, our proposed confirmation conspire is lightweight & free coupling to concrete secure inquiry plots & can be effectively prepared into any protected question conspire for distributed computing. In any case, some essential augmentations & imperative works should be additionally provided to culminate our unique plan, for example, itemized execution assessment & formal security definition & verification. All the more essentially, in the unscrupulous cloud condition, the plan experiences the accompanying two imperative security issues: 1) Just as conceivably altering or erasing question comes about, the untrustworthy cloud server may likewise alter or manufacture confirmation objects themselves to make the information client difficult to perform check operation. Uncommonly, once the cloud server realizes that the inquiry comes about check conspire is given in the safe pursuit framework, he may return inveracious confirmation protest escape obligations of rowdiness. 2) When an information client needs to get the coveted confirmation protest, some imperative data will be uncovered, for example, which check objects are being or have been asked for before much of the time, & so forth. This data may spill question client's protection & uncover some helpful substance about information records. All the more significantly, this uncovered data may progress toward becoming allurements of trouble making for the cloud server.

**1.1 Objective of the Project**

Secure search strategies over scrambled cloud information enable an approved client to inquiry information documents of enthusiasm by submitting encoded question catchphrases to the cloud server in a protection saving way. Notwithstanding, practically speaking, the returned question results might be wrong or deficient in the untrustworthy cloud condition. For instance, the cloud server may purposefully discard some qualified outcomes to spare computational assets & correspondence overhead. In this manner, a well-working secure question framework ought to give an inquiry comes about confirmation instrument that enables the information client to check comes about. In this paper, we plan a safe, effectively incorporated, & fine-grained question comes about confirmation system, by which, given a scrambled inquiry comes about set, the inquiry client not exclusively can check the rightness of every datum document in the set yet additionally can additionally check what number of or which qualified information records are not returned if the set is deficient before decoding. The confirmation plot is free coupling to concrete secure hunt methods & can be effectively incorporated into any safe question conspire. We accomplish the objective by building secure confirmation protest for encoded cloud information. Besides, a short mark procedure with amazingly little stockpiling cost is proposed to ensure the validness of confirmation question & a check protest ask for system is exhibited to enable the inquiry client to safely get the coveted confirmation question. Execution assessment demonstrates that the proposed plans are functional & effective.

**2. LITERATURE SURVEY**

**Security Challenges for the Public Cloud**

In this discussion, we will initially talk about various squeezing security challenges in Cloud Computing, including information benefit outsourcing security & secure calculation outsourcing. At that point, we will concentrate on information stockpiling security in Cloud Computing. As one of the primitive administrations, distributed storage enables information proprietors to outsource their information to cloud for its engaging advantages. Nonetheless, the way that proprietors never again have physical ownership of the outsourced information raises huge security worries on the capacity accuracy.

Thus, empowering secure capacity examining in the cloud condition with new methodologies winds up noticeably basic & testing In this discussion, we will introduce our current research endeavors towards capacity outsourcing security in distributed computing & depict both our specialized methodologies & security & execution assessments.

**Pragmatic Techniques for Searches on Encrypted Data**

It is attractive to store information on information stockpiling servers, for example, mail servers & document servers in scrambled shape to decrease security & protection dangers. Yet, this more often than not infers that one needs to give up usefulness for security. For instance, if a customer wishes to recover just reports containing certain words, it was not already known how to let the information stockpiling server play out the pursuit & answers the inquiry without loss of information secrecy. In this paper, we depict our cryptographic plans for the issue of looking on encoded information & give evidences of security to the subsequent crypto frameworks. Our procedures have various significant favorable circumstances. They are provably secure: they give provable mystery to encryption, as in the untrusted server can't get the hang of anything about the plaintext when just given the ciphertext; they give inquiry detachment to looks, implying that the untrusted server can't pick up much else about the plaintext than the query item; they give controlled seeking, so that the untrusted server can't scan for a discretionary word without the client's approval; they likewise bolster concealed questions, so the client may approach the untrusted server to look for a mystery word without uncovering the word to the server.

We have depicted new procedures for remote looking on scrambled information utilizing an untrusted server & gave evidences of security to the subsequent crypto frameworks. Our systems have various critical focal points: they are provably secure; they bolster controlled & concealed inquiry & question segregation; they are straightforward & quick (More particularly, for a report of length , the encryption & pursuit calculations just need O(n) stream figure & square figure operations); & they present no space & correspondence overhead. Our plan is likewise extremely adaptable, & it can without much of a stretch be reached out to help further developed hunt questions. We presume this gives an effective new building hinder for the development of secure administrations in the untrusted foundation.

**Open Key Encryption with keyword Search**

We consider the issue of looking on data that is encoded using an open key structure. Consider customer Bob who sends email to customer Alice encoded under Alice's open key. An email entryway needs to test whether the email contains the watchword "basic" with the objective that it could course the email in like way. Alice, of course does not wish to give the entry the ability to decipher each one of her messages. We describe & manufacture a framework that enables Alice to give a key to the entryway that engages the gateway to test whether "sincere" is a watchword in the email without getting the hang of whatever else about the email. We insinuate this segment as Public Key Encryption with catchphrase Search. As another representation, consider a mail server that stores diverse messages straightforwardly encoded for Alice by others. Using our segment Alice can send the mail server a key that will engage the server to perceive all messages containing some specific catchphrase, yet get the hang of nothing else. We describe open key encryption with watchword request & give a couple of improvements.

We described the possibility of an open key encryption with catchphrase look (PEKS) & gave two advancements. Building up a PEKS is related to Identity Based Encryption (IBE), however PEKS is from every angle harder to assemble. We showed that PEKS recommends Identity Based Encryption, however the inverse is starting at now an open issue. Our improvements for PEKS rely upon late IBE advancements.We can demonstrate security by misusing additional properties of these plans.

**Accessible Symmetric Encryption: Improved Definitions & Efficient Constructions**

Accessible symmetric encryption (SSE) enables a gathering to outsource the capacity of his information to another gathering in a private way, while keeping up the capacity to specifically look over it. This issue has been the concentration of dynamic research & a few security definitions & developments have been proposed. In this paper we start by looking into existing thoughts of security & propose new & more grounded security definitions. We at that point display two developments that we indicate secure under our new definitions. Strikingly, notwithstanding fulfilling more grounded security ensures, our developments are more productive than every single past development. Further, earlier work on SSE just considered the setting where just the proprietor of the information is equipped for submitting look inquiries. We consider the regular expansion where a discretionary gathering of gatherings other than the proprietor can submit look questions. We formally characterize SSE in this multi-client setting, & present an effective development.

In this article, we have returned to the issue of accessible symmetric encryption, which enables a customer to store its information on a remote server such that it can look over it in a private way. We make a few commitments including new security definitions & new developments. Propelled by inconspicuous issues in all past security definitions for SSE, we propose new definitions & call attention to that the current ideas have huge down to earth disadvantages: in spite of the characteristic utilization of accessible encryption, they just assurance security for clients that play out all their pursuits without a moment's delay. We address this impediment by presenting more grounded definitions that certification security notwithstanding when clients perform more practical ventures. We likewise propose two new SSE developments. Shockingly, in spite of being provably secure under our more grounded security definitions, these are the most effective plans to date & are (asymptotically) ideal (i.e., the work performed by the server per returned record is steady in the span of the information). At last, we additionally consider multi-client SSE, which stretches out the looking capacity to parties other than the proprietor.

**Deterministic & Efficiently Searchable Encryption**

We present as-solid as-conceivable meanings of security, & developments accomplishing them, for open key encryption plans where the encryption calculation is deterministic. We get as an outcome database encryption techniques that allow quick (i.e. sub-straight, & in actuality logarithmic, time) look while provably giving protection that is as solid as conceivable subject to this quick pursuit imperative.

One of our builds, called RSA-DOAEP, has the additional element of being length safeguarding, with the goal that it is the primary case of an open key figure. We sum up this to acquire an idea of productively accessible encryption plans which allow more adaptable protection to seek time exchange offs by means of a method called bucketization. Our outcomes answer muchasked inquiries in the database group & give establishments to work done there.

**Open Key Encryption with Fuzzy Keyword Search: A Provably Secure Scheme under Keyword Guessing Attack**

A considerable measure of intrigue has been drawn as of late into open key encryption with catchphrase seek (PEKS), which keeps publickey scrambled archives amendable to secure watchword look. In any case, PEKS oppose against watchword speculating assault by accepting that the extent of the catchphrase space is past the polynomial level. Be that as it may, this suspicion is incapable practically speaking. PEKS are unreliable under watchword speculating assault. As we watch, the way to shield such assault is to maintain a strategic distance from the accessibility of the correct hunt trapdoor to foes. In like manner, we trade off the precision of inquiry trapdoor by mapping no less than two distinct catchphrases into a fluffy hunt trapdoor. We propose a novel idea called open key encryption with fluffy watchword look (PEFKS), by which the un-trusted server just gets the fluffy hunt trapdoor rather than the correct pursuit trapdoor, & characterize its semantic security under picked catchphrase assault (SS-CKA) & lack of definition of catchphrases under non-adaptively picked watchwords & watchword speculating assault (IK-NCK-KGA). For the watchword space with & without uniform circulation, we individually show two general changes from mysterious character based encryption to PEFKS, & demonstrate their SSCKA & IK-NCK-KGA securities. As far as anyone is concerned, PEFKS is the main plan to oppose against catchphrase speculating assault on condition that the watchword space isn't more than the polynomial level.

In PEKS, an intermediary server, who reacts the catchphrase questions of a collector, can know the substance of watchwords by actualizing KGA. In addition, it is effective under the down to earth condition that the measure of the watchword space isn't more than the polynomial level. So as to oppose against KGA, we novelly characterized open key encryption with fluffy catchphrase look (PEFKS) & its IK-NCK-KGA security. Also, we proposed two all inclusive changes from IBE to PEFKS under various conditions. Under the condition that the watchword space has uniform dissemination, we proposed a SS-CKA & IK-NCK-KGA secure change PEFKS-UD, & gave an occurrence in light of BF01 conspire. Under the condition that the catchphrase space has non-uniform conveyance, we proposed another SS-CKA & IK-NCK-KGA secure change PEFKS-ND, & gave two strategies to sort watchwords, which is the way to acknowledge PEFKS-ND. Past the viewpoint of cryptosystem, we talked about the one-sided favorable position of KGA on PEFKSND, which is caused just by the non-uniform dissemination of the watchword space. We enlighten that the one-sided advantage has been diminished however much as could be expected. So we influenced PEFKS-ND to secure in an expansive sense.

**Parallel & Dynamic Searchable Symmetric Encryption**

Accessible symmetric encryption (SSE) empowers a customer to outsource a gathering of scrambled archives in the cloud & hold the capacity to perform catchphrase seeks without uncovering data about the substance of the reports & inquiries. Albeit effective SSE developments are known, past arrangements are exceptionally successive. This is fundamentally because of the way that, as of now, the main strategy for accomplishing sub-direct time look is the reversed file approach which requires the hunt calculation to get to a grouping of memory areas, each of which is unusual & put away at the past area in the arrangement. Propelled by progresses in multi-center models, we display another strategy for developing sub-straight SSE plans. Our approach is exceptionally parallelizable & dynamic. With about a logarithmic number of centers set up, scans for a catchphrase w in our plan execute in o(r) parallel time, where r is the quantity of archives containing watchword w (with more centers, this bound can go down to O(log n), i.e., autonomous of the outcome measure r). Such time unpredictability outflanks the ideal Θ(r) consecutive hunt time—a comparative headed holds for the updates. Our plan likewise accomplishes the accompanying critical properties: (an) it appreciates a solid idea of security, to be specific security against versatile picked catchphrase assaults; (b) contrasted with existing sub-straight unique SSE plans, refreshes in our plan don't release any data, aside from data that can be gathered from past pursuit tokens; (c) it can be actualized effectively in outside memory (with logarithmic I/O overhead). Our strategy is straightforward & utilizes a red-dark tree information structure; its security is demonstrated in the irregular prophet show.

**Dynamic Searchable Encryption by means of Blind Storage**

Dynamic Searchable Symmetric Encryption enables a customer to store a dynamic accumulation of scrambled reports with a server, & later rapidly do watchword looks on these encoded archives, while uncovering insignificant data to the server. In this paper we exhibit another dynamic SSE plot that is less difficult & more effective than existing plans while uncovering less data to the server than earlier procedure, accomplishing entirely versatile protection against legitimate yet probing servers. We actualized a model of our preparation & showed its proficiency on datasets from earlier work. Aside from its solid effectiveness, our plan is additionally less complex: specifically, it doesn't require the server to help any operation other than transfer & download of information. In this manner the server in our plan can be construct exclusively with respect to a distributed storage benefit, instead of a cloud calculation benefit too, as in earlier work. In building our dynamic SSE conspire, we present another primitive called Blind Storage, which enables a customer to store an arrangement of records on a remote server such that the server does not figure out what number of documents are put away, or the lengths of the individual documents; as each record is recovered, the server finds out about its reality (and can see a similar record being downloaded therefore), yet the record's name & substance are not uncovered. This is a primitive with a few applications other than SSE, & is of autonomous intrigue.

In this work, we presented another cryptographic build called Blind Storage, & actualized it utilizing a novel, yet light-weight convention SCATTERSTORE. We likewise indicated how a dynamic SSE plan can be built utilizing Blind Storage, in a moderately straightforward way. The subsequent plan is all the more computationally effective, require less difficult foundation, & is more secure than the current plans.

**Secure Ranked Keyword Search over Encrypted Cloud Data**

As distributed processing ends up noticeably common, delicate data are as a rule progressively concentrated into the cloud. For the assurance of information security, touchy information must be encoded before outsourcing, which makes powerful information use an exceptionally difficult errand. Albeit customary accessible encryption plans enable clients to safely look over encoded information through catchphrases, these systems bolster just boolean inquiry, without catching any importance of information records. This approach experiences two fundamental disadvantages when specifically connected with regards to Cloud Computing. From one viewpoint, clients, who don't really have pre-information of the encoded cloud information, need to postprocess each recovered record keeping in mind the end goal to discover ones most coordinating their enthusiasm; On the other hand, perpetually recovering all documents containing the questioned watchword additionally acquires superfluous system movement, which is completely unwanted in the present pay-as-you-utilize cloud worldview. In this paper, out of the blue we characterize & tackle the issue of powerful yet secure positioned catchphrase seek over scrambled cloud information. Positioned look enormously improves framework convenience by restoring the coordinating records in a positioned arrange with respect to certain importance criteria (e.g., watchword recurrence), in this way making one bit nearer towards handy organization of security safeguarding information facilitating administrations in Cloud Computing. We initially give a clear yet perfect development of positioned catchphrase seek under the best in class accessible symmetric encryption (SSE) security definition, & exhibit its wastefulness. To accomplish more down to earth execution, we at that point propose a definition for positioned accessible symmetric encryption, & give a proficient plan by appropriately using the current cryptographic primitive, arrange saving symmetric encryption (OPSE). Intensive examination demonstrates that our proposed arrangement appreciates "as-solid as could reasonably be expected" security ensure contrasted with past SSE plans, while effectively understanding the objective of positioned catchphrase look. Broad trial comes about exhibit the effectiveness of the proposed arrangement.

In this paper, as an underlying endeavor, we persuade & tackle the issue of supporting proficient positioned catchphrase scan for accomplishing powerful use of remotely put away encoded information in Cloud Computing. We initially give an essential plan & demonstrate that by following the same existing accessible encryption system, it is extremely wasteful to accomplish positioned look. We at that point suitably debilitate the security ensure, fall back on the recently generated crypto primitive OPSE, & infer a productive one-to-many request protecting mapping capacity, which permits the compelling RSSE to be planned. Through exhaustive security investigation, we demonstrate that our proposed arrangement is secure & protection safeguarding, while effectively understanding the objective of positioned watchword seek.

**Security Preserving Multi-Keyword Ranked Search with Anonymous ID Assignment over Encrypted Cloud Data**

The progress in disseminated processing has propelled the data proprietors to outsource their data organization systems from adjacent districts to business open cloud for phenomenal flexibility and monetary hold reserves.However, individuals can appreciate full advantage of distributed computing in the event that we can address genuine protection & security worries that accompany putting away touchy individual data. For genuine security, client character ought to stay escaped CSP (Cloud specialist organization) & to ensure protection of information, information which is touchy is to be scrambled before outsourcing. In this way, empowering a scrambled cloud information seek benefit is of extraordinary significance. By considering the substantial number of information clients, reports in the cloud, it is imperative for the inquiry administration to permit multikeyword question & give result closeness positioning to meet the compelling need of information recovery seek & not regularly separate the indexed lists. In this framework, we characterize & tackle the testing issue of protection safeguarding multikeyword positioned look over scrambled cloud information (MRSE), & set up an arrangement of strict protection necessities for such a safe cloud information usage framework to be executed in genuine. We initially propose a fundamental thought for the Multi-catchphrase Ranked Search over Encrypted cloud information (MRSE) in view of secure internal item calculation & proficient closeness measure of organize coordinating, i.e., however many matches as would be prudent, keeping in mind the end goal to catch the importance of information archives to the hunt question, at that point we give two essentially enhanced MRSE plans to accomplish different stringent protection prerequisites in two diverse risk models. Task of mysterious ID to the client to give greater security to the information on cloud server is finished. To enhance the inquiry experience of the information seek benefit, advance augmentation of the two plans to help more hunt semantics is finished.

The past work for the most part centered around giving security to the information on cloud in which utilizing multi-catchphrase positioned look was given over encoded cloud information utilizing productive comparability measure of co-ordinate coordinating. The past work likewise proposed a fundamental thought of MRSE utilizing secure internal item calculation. There was a need to give all the more genuine security which this paper presents. In this framework, stringent security is given by allocating the cloud client a special ID. This client ID is kept escaped the cloud specialist co-op & also the outsider client to shield the client's information on cloud from the CSP & the outsider client. Therefore, by concealing the client's personality, the secrecy of client's information is kept up.

**3. ANALYSIS**

**3.1 Existing System**

As of late, with the developing prevalence of distributed computing, how to safely & proficiently seek over scrambled cloud information turns into an exploration center. Some methodologies have been proposed in view of customary accessible encryption plans, which mean to ensure information security & inquiry protective measures with better question effective for distributed computing. In any case, these plans depend on a perfect suspicion that the cloud server is a "fair however inquisitive" element & keeps vigorous & secure programming/equipment conditions. Thus, rectify & finish inquiry comes about dependably are unexceptionally come back from the cloud server when a question closes without fail. In any case, in reasonable applications, the cloud server may return mistaken or inadequate question comes about once he carries on insincerely for illicit benefits, for example, sparing calculation & correspondence cost or because of conceivable programming/equipment disappointment of the server.

**Disadvantages of Existing System:**

1. The check components give a coarse-grained confirmation, i.e., if the inquiry result set contains all qualified & right information records, at that point these plans answer yes, generally answer no. Hence, if the confirmation calculation yields no, an information client needs to prematurely end the decoding for all question comes about in spite of just a single inquiry result is mistaken.
2. The check components are for the most part firmly coupled to relating secure question developments & have not all inclusiveness

**3.2 Proposed System**

We formally propose the certain protected inquiry framework show & risk demonstrate & outline a fine-grained question comes about check plot for secure watchword seek over scrambled cloud information. We propose a short mark system in view of certificateless open key cryptography to ensure the validness of the check objects themselves. We plan a novel check protest ask for system in view of Paillier Encryption, where the cloud server knows nothing about what the information client is asking for & which confirmation objects are come back to the client.

**Advantages of Proposed System:**

1. Our plan can check the rightness of each scrambled inquiry result or further precisely discover what number of or which qualified information records are returned by the untrustworthy cloud server
   1. **SYSTEM REQUIREMENTS**

**Hardware Requirements:**

* Processor - Pentium –IV
* Speed - 1.1 GHz
* RAM - 256 MB(min)
* Hard Disk - 20 GB

**Software Requirements:**

* Operating System - Windows XP/95/98/2000
* Programming Language - JAVA, jdk1.6
* Front End - AWT, Swing
* Database - MySQL

**4. DESIGNING**

**UML Diagrams:**

The Unified Modeling Language (UML) is provide the users software blueprint & also called as “Diagrams”.

The UML diagrams are basically two types;

1. Structural Diagrams
2. Behavioral Diagrams

And also, we have 4 types of relationships are there to connect the classes, & objects to each other in the UML.

The following Relationships are:

1. Association (Represented by )
2. Aggregation (Represented by )
3. Generalization (Represented by )
4. Dependency (Represented by )

**4.1 Class Diagram:**



**4.2 Use Case Diagram:**

****

**4.3 Sequence Diagram:**



**4.4 Collaboration Diagram:**



**4.5 Component Diagram:**



**4.6 Deployment Diagram:**



**4.7 Activity Diagram:**

Secure Search System

Cloud Server

Data Owner

Start Server

Register

Save Uploaded files

Is he already registered?

No

Yes

Generate verification object

Login

Upload Files

Search Query

Verification

Download & Decrypt file

Logout

View chart

**Data Flow Diagram (DFD):**

This is not a UML diagram & it is the context diagram. In this Diagram we can represents our complete system structure & system work flow.

5. Download & decrypt the file

3. Upload files

Cloud Server

Data Owner

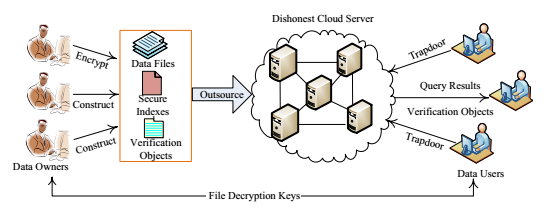
2. Register & login into system

4. Search queries & verification

6. View chart 1. Run

**5. IMPLEMENTATION**

**System Architecture**



**5.1 MODULES**

1. Data Owner Module
2. Data User Module
3. Cloud Server Module

**Module Description:**

**Data Owner:**

Data owners encrypt their private data & upload them to cloud server for enjoying the abundant benefits brought by the cloud computing as well as guaranteeing data security. Meanwhile, the secure searchable indexes are also constructed to support effective keyword search over encrypted outsourced data.

**Data User:**

An authorized data user obtains interested data files from the cloud server by submitting query trapdoors (encrypted query keywords) to the cloud server.

**Cloud Server:**

Cloud server can performs search over secure indexes according to trapdoors & sends the query results to the data user.

**5.2 TECHNOLOGY**

To Develop this project we used technology is Java.

**About Java**:

• Java is a software engineer's dialect

• Java is firm & reliable

• Besides for those limitation forced by method for the web condition. Java gives the software engineer, full control

In the long run, Java is to web Programming the place C was once to approach Programming.

**Significance of Java to the Web**

Java affected the web. That is because of the way that; java extends the Universe of items that may move about openly in the internet. In a system, two classes of articles are transmitted between the server & the private PC. They are inactive ability & Dynamic fiery applications. Yet, Java settles these worries & by method for doing as such, has opened the way to a novel new type of utilization called the Applet.

**Java Architecture**

Java design gives a moveable, viable, over the top performing climate for advance. Java gives convey ability through ordering the byte codes for the JVM, which is then translated on every last stage by utilizing the run-time condition. Java is a dynamic framework, prepared to stack code when required from a registering gadget in a similar room or all through the planet.

**Compilation of code:**

Whilst you compile the code, the Java compiler generates computing device code (called byte code) for a hypothetical machine named Java Virtual Machine (JVM). The JVM is meant t completed the byte code. The JVM is generated for the resolving the drawback of probability. The code is written & compiled for one system & interpreted on all machines .This computing device is known as Java virtual Machine (JVM).

**Compiling & interpreting java source code:**

**Source code**

**Pc compiler**

**Macintosh compiler**

**SPARC Compiler**

**Java Byte code**

**Platform independent**

**Java interpreter**

**Java interpreter**

**macintosh**

**)))**

**Java interpreter(SPARC)**

For the period of run-time the Java interpreter tricks the byte code file into thinking that it's strolling on a Java digital desktop. Surely this could be Intel Pentium windows ninety five or solar SPARCstation strolling Solaris or Apple Macintosh walking procedure & all might obtain code from any pc by means of internet & run the Applets.

**Simple**:

Java was once designed to be effortless for the official programmer to be taught & to use with ease. If you are a skilled C++ Programmer, learning Java will oriented points of C++. Many of the confusing standards from C++ are both left out of Java friendlier method. In Java there is a small quantity of obviously outlined methods to achieve a given challenge.

### Object oriented

Java was once now not designed to be supply-code compatible with some other language. This allowed the Java staff the freedom to design with a clean state. One outcome of this was a smooth usable, pragmatic process to objects. The thing mannequin in Java is simple & convenient to extend, while simple types, similar to integers, are stored as high-efficiency non-objects.

### Robust

The multi-platform atmosphere of the web areas wonderful demands on software, on account that the software need to execute reliably in a form of programs. The capacity to generate powerful packages was given an excessive precedence in the design of Java? Java is exactly typed language; it assessments your code at assemble time & runtime.

Java nearly eliminates the problems of reminiscence administration & deal area, which is entirely automatic. In a good-written Java program, all run-time error can & must be managed by way of your program.

**5.3 Sample Code**

**UserScreen.java**

package com;

import javax.swing.\*;

import java.awt.\*;

import java.net.Socket;

import java.io.\*;

import java.util.\*;

import java.math.BigInteger;

import org.jfree.ui.RefineryUtilities;

public class UserScreen extends JFrame{

JButton b1,b2,b3,b4,b5;

JPanel p1,p2;

Font f1;

JTextArea area;

JScrollPane jsp;

Login login;

String user;

JFileChooser chooser;

ArrayList<byte[]> trapdoor = new ArrayList<byte[]>();

ArrayList<VerificationObject> vlist;

long time;

int keyword\_size;

public UserScreen(Login log,String usr){

super("User Screen");

login = log;

user = usr;

p1 = new JPanel();

f1 = new Font("Monospaced",Font.BOLD,16);

chooser = new JFileChooser();

b1 = new JButton("Upload File");

b1.setFont(f1);

p1.add(b1);

int option = chooser.showOpenDialog(UserScreen.this);

if(option == JFileChooser.APPROVE\_OPTION){

File file = chooser.getSelectedFile();

upload(file);

}

}

});

b2 = new JButton("Search Query");

b2.setFont(f1);

p1.add(b2);

search();

}

});

b5 = new JButton("Verification");

b5.setFont(f1);

p1.add(b5);

verification();

}

});

b3 = new JButton("Keywords Vs Verification");

b3.setFont(f1);

p1.add(b3);

Chart chart1 = new Chart("Keywords Vs Verification Chart",time,keyword\_size);

chart1.pack();

RefineryUtilities.centerFrameOnScreen(chart1);

chart1.setVisible(true);

}

});

b4 = new JButton("Logout");

b4.setFont(f1);

p1.add(b4);

setVisible(false);

login.setVisible(true);

}

});

p2 = new JPanel();

p2.setLayout(new BorderLayout());

area = new JTextArea();

area.setFont(f1);

area.setEditable(false);

jsp = new JScrollPane(area);

p2.add(jsp,BorderLayout.CENTER);

getContentPane().add(p1,BorderLayout.NORTH);

getContentPane().add(p2,BorderLayout.CENTER);

PaillierEnc.KeyGeneration();

}

public void verification(){

try{

long start = System.currentTimeMillis();

ArrayList<String> hash = new ArrayList<String>();

ArrayList<String> bloom = new ArrayList<String>();

for(int i=0;i<trapdoor.size();i++){

BloomFilter.generateBloom(50,trapdoor.get(i));

StringBuilder sb = new StringBuilder();

for(int j=0;j<BloomFilter.input.length;j++){

sb.append(BloomFilter.input[j]);

}

System.out.println(sb.toString()+" "+new String(AES.decrypt(trapdoor.get(i))));

bloom.add(sb.toString());

}

for(int i=0;i<bloom.size();i++){

BigInteger input = new BigInteger(bloom.get(i).getBytes());

hash.add(PaillierEnc.Encryption(input).toString());

}

StringBuilder sb = new StringBuilder();

for(int i=0;i<hash.size();i++){

for(int j=0;j<vlist.size();j++){

VerificationObject vo = vlist.get(j);

keyword\_size = keyword\_size + vo.getKeyword().size();

for(int k=0;k<vo.getHash().size();k++){

String h = vo.getHash().get(k);

if(h.equals(hash.get(i))){

File ff = new File(vo.getPath());

File ff1 = new File(ff.getParent());

sb.append(ff1.getName()+"/"+ff.getName()+",");

}

}

}

}

System.out.println("verify "+sb.toString());

if(sb.length() > 0){

sb.deleteCharAt(sb.length()-1);

ViewSearchResult vsr = new ViewSearchResult();

String arr1[] = sb.toString().trim().split(",");

for(int i=0;i<arr1.length;i++){

String row[] = {arr1[i]};

vsr.dtm.addRow(row);

}

vsr.setVisible(true);

vsr.setSize(600,400);

vsr.setTitle("Files obtains from verification object for same query");

}else{

JOptionPane.showMessageDialog(this,"No record found");

}

long end = System.currentTimeMillis();

time = end - start;

}catch(Exception e){

e.printStackTrace();

}

}

public void upload(File file){

try{

FileInputStream fin = new FileInputStream(file);

byte file\_data[] = new byte[fin.available()];

fin.read(file\_data,0,file\_data.length);

fin.close();

byte enc[] = AES.encrypt(file\_data);

String keywords = new String(file\_data);

String arr[] = keywords.trim().toLowerCase().split("\\s+");

ArrayList<byte[]> encrypted\_keywords = new ArrayList<byte[]>();

ArrayList<String> bloom = new ArrayList<String>();

ArrayList<String> hash = new ArrayList<String>();

ArrayList<String> dup = new ArrayList<String>();

for(int i=0;i<arr.length;i++){

if(!dup.contains(arr[i])){

dup.add(arr[i]);

byte data[] = AES.encrypt(arr[i].getBytes());

encrypted\_keywords.add(data);

}

}

for(int i=0;i<encrypted\_keywords.size();i++){

BloomFilter.generateBloom(50,encrypted\_keywords.get(i));

StringBuilder sb = new StringBuilder();

for(int j=0;j<BloomFilter.input.length;j++){

sb.append(BloomFilter.input[j]);

}

System.out.println(sb.toString()+" "+new String(AES.decrypt(encrypted\_keywords.get(i))));

bloom.add(sb.toString());

}

for(int i=0;i<bloom.size();i++){

BigInteger input = new BigInteger(bloom.get(i).getBytes());

hash.add(PaillierEnc.Encryption(input).toString());

}

Socket socket = new Socket("localhost",3333);

ObjectOutputStream out = new ObjectOutputStream(socket.getOutputStream());

ObjectInputStream in = new ObjectInputStream(socket.getInputStream());

Object req[]={"upload",user,file.getName(),enc,encrypted\_keywords,bloom,hash};

out.writeObject(req);

out.flush();

Object res[]=(Object[])in.readObject();

String response = (String)res[0];

area.append(response+"\n");

out.close();

in.close();

socket.close();

}catch(Exception e){

e.printStackTrace();

}

}

public void search(){

try{

String input = JOptionPane.showInputDialog(this,"Enter input query");

if(input != null){

input = input.trim().toLowerCase();

String arr[] = input.split("\\s+");

trapdoor.clear();

for(int i=0;i<arr.length;i++){

byte enc[] = AES.encrypt(arr[i].trim().getBytes());

trapdoor.add(enc);

area.append("Query = "+arr[i]+" Encrypted trapdoor = "+new String(enc)+"\n");

}

Socket socket = new Socket("localhost",3333);

ObjectOutputStream out = new ObjectOutputStream(socket.getOutputStream());

ObjectInputStream in = new ObjectInputStream(socket.getInputStream());

Object req[]={"query",user,trapdoor};

out.writeObject(req);

out.flush();

Object res[]=(Object[])in.readObject();

String response = (String)res[0];

System.out.println("==="+response);

vlist = (ArrayList<VerificationObject>)res[1];

if(!response.equals("No record found for given querys")){

ViewSearchResult vsr = new ViewSearchResult();

String arr1[] = response.split(",");

int random = getRandom();

int size = arr1.length;

if(random == 1)

size = size - 1;

for(int i=0;i<size;i++){

String row[] = {arr1[i]};

vsr.dtm.addRow(row);

}

vsr.setVisible(true);

vsr.setSize(600,400);

}else{

JOptionPane.showMessageDialog(this,response);

}

}

}catch(Exception e){

e.printStackTrace();

}

}

public int getRandom(){

Random r = new Random();

return r.nextInt(2);

}

}

**BloomFilter.java**

package com;

import java.util.ArrayList;

public class BloomFilter{

static byte encrypt\_data[];

static int input[] = new int[50];

static ArrayList<Integer> list = new ArrayList<Integer>();

public static void generateBloom(int key,byte enc[]){

encrypt\_data = enc;

list.clear();

for(int i=0;i<encrypt\_data.length;i++){

String data = new String(Byte.toString(encrypt\_data[i]));

int bloom = data.hashCode()%key;

list.add(bloom);

}

for(int i=0;i<50;i++){

if(list.contains(i))

input[i] = 1;

else

input[i] = 0;

}

}

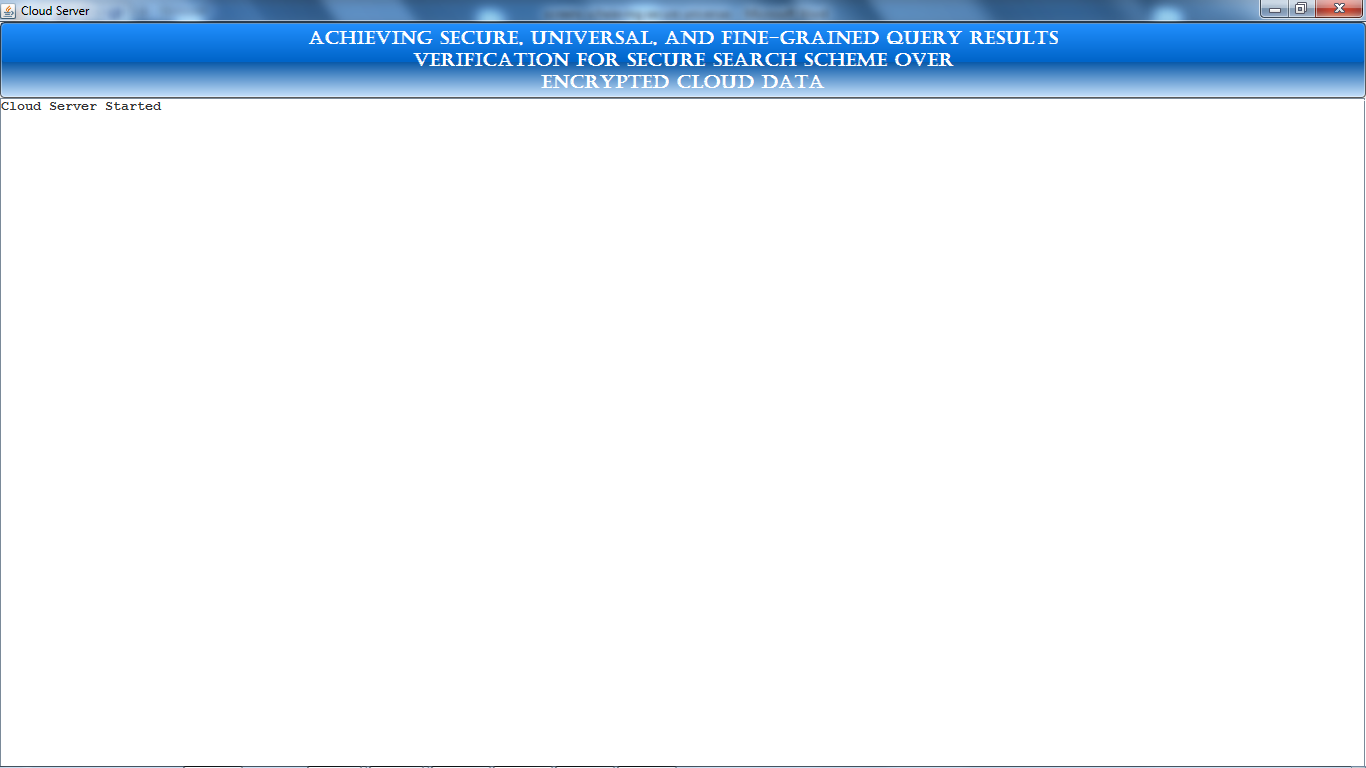
}

**6. TEST CASES**

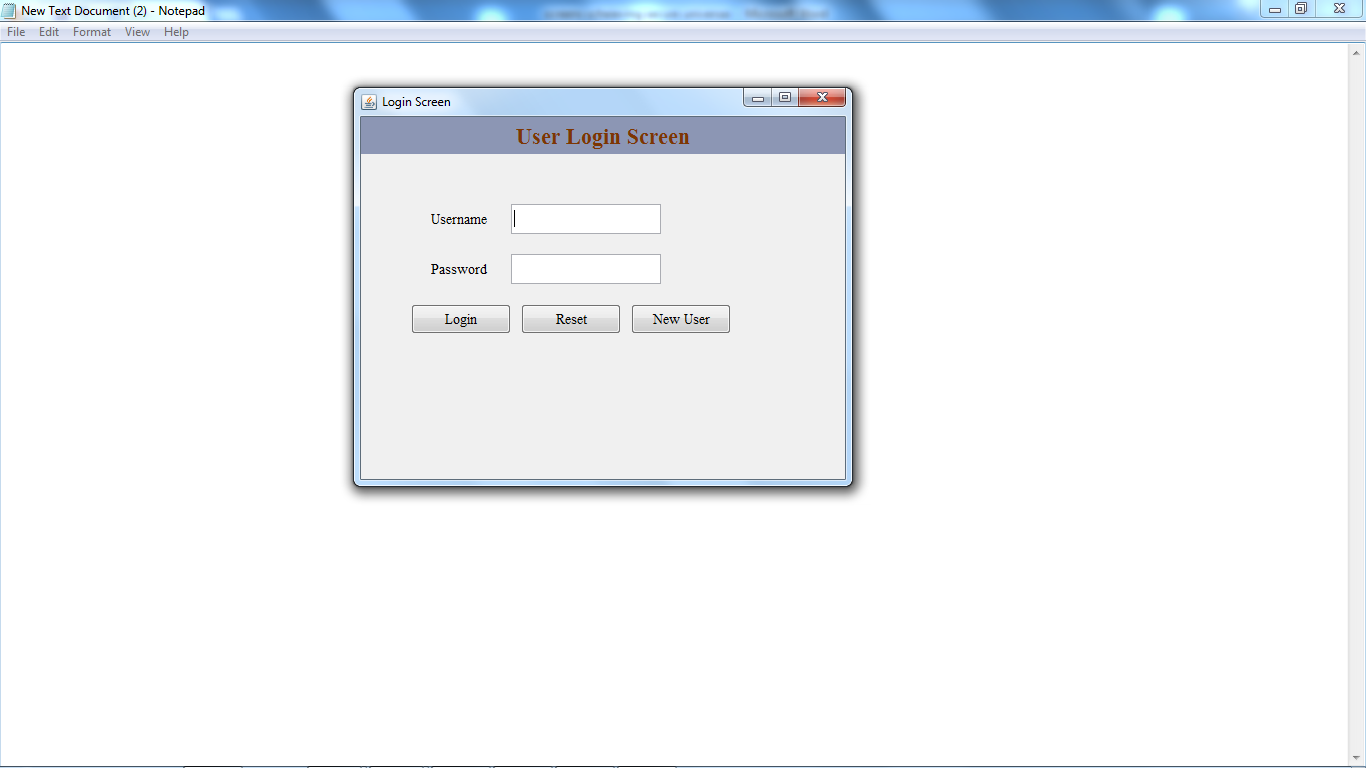
|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Test Case Id** | **Test Case Name** | **Test Case Desc.** | **Test Steps** | | | **Test Case Status** | **Test Priority** |
| **Step** | **Expected** | **Actual** |
| 01 | Start Server | Test whether the cloud server will started or not | If server not started | Then no file will be stored in the cloud server | Server started | High | High |
| 02 | Login | Verify the Data Owner login | Without register | data owner cannot login into the system | Display the data owner home page | High | High |
| 03 | Upload file | Verify the either file uploaded or not | If data owner not upload the file | Then no search results will be displayed | Uploaded successfully & saved in cloud server | High | High |
| 04 | Verification | Test whether the verification will be done or not | Without verification object | Cloud server cannot verify the file | Decrypt & download the file & file saved in system directory | High | High |

**7. SCREEN SHOTS**

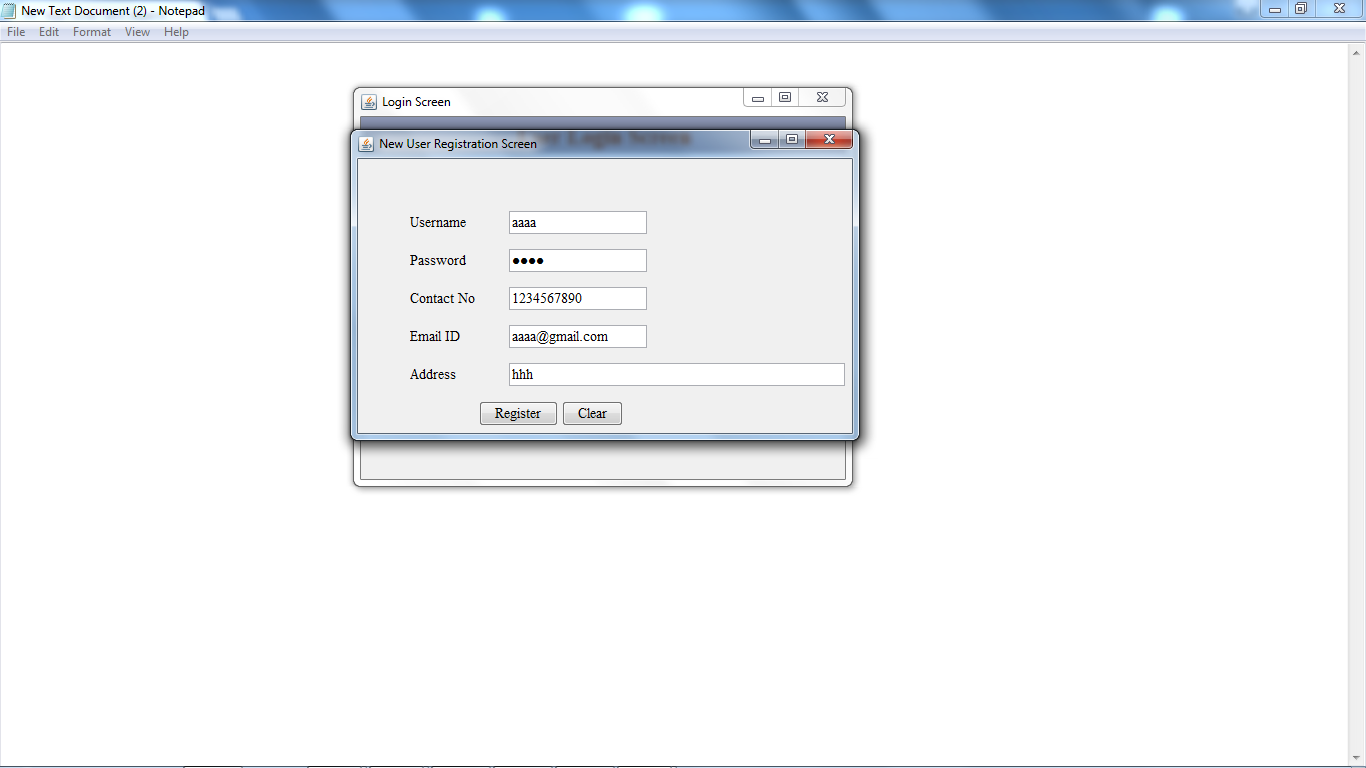
Cloud server:



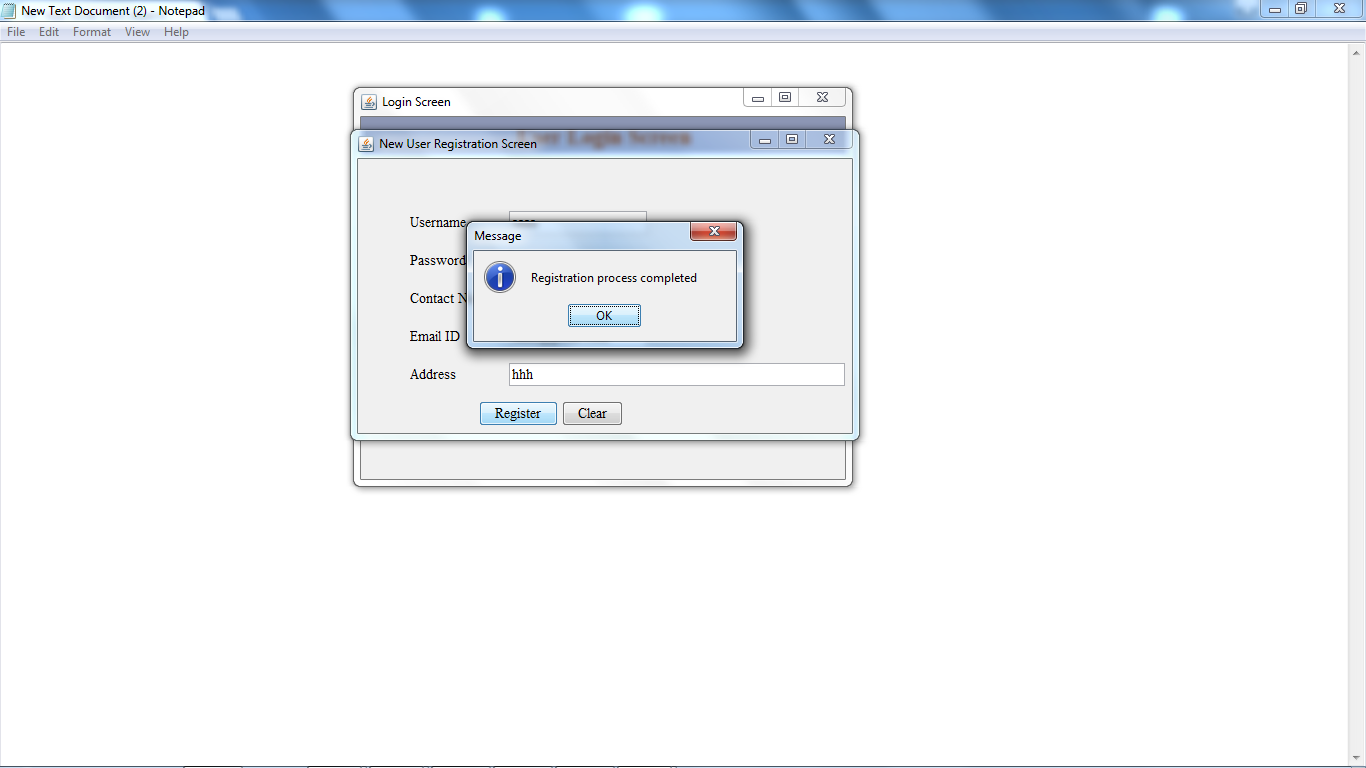
Data owner/ user welcome screen:



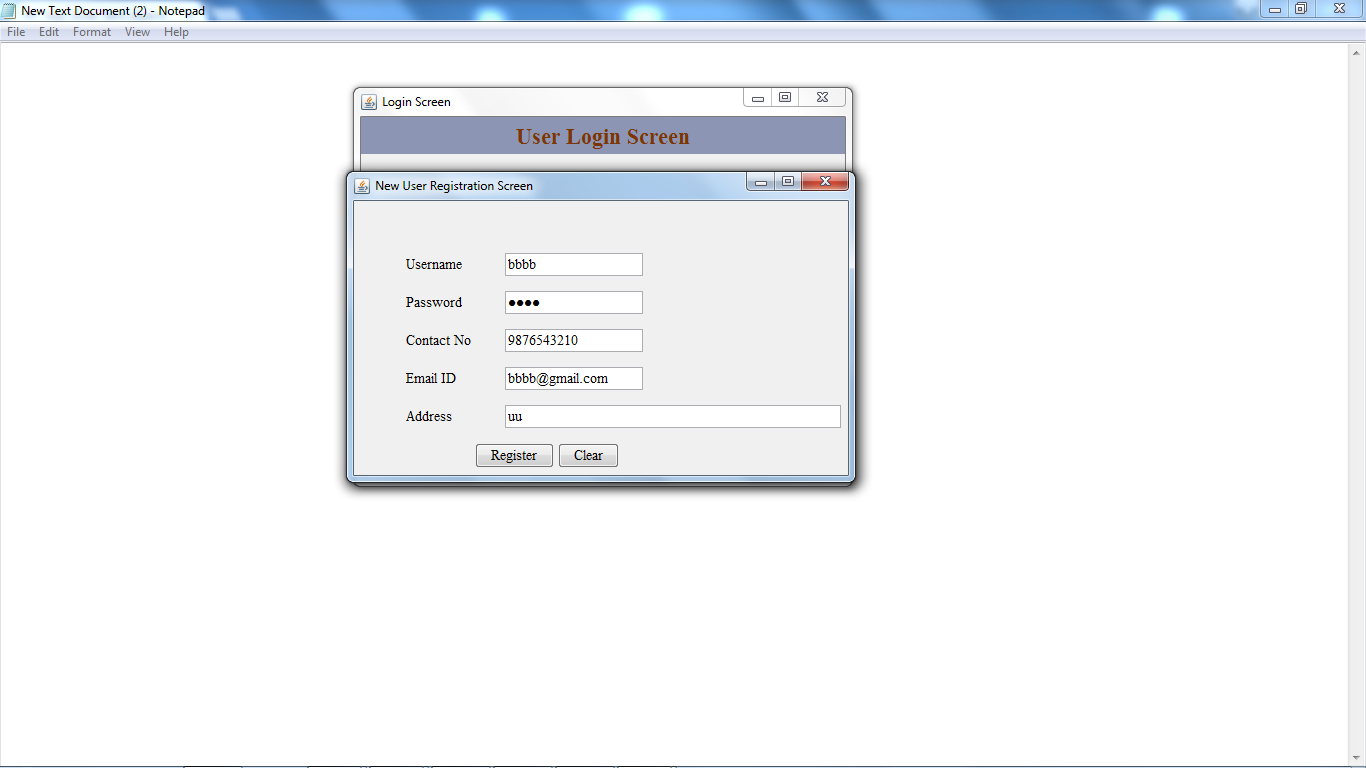
Click on new user to register a data owner:



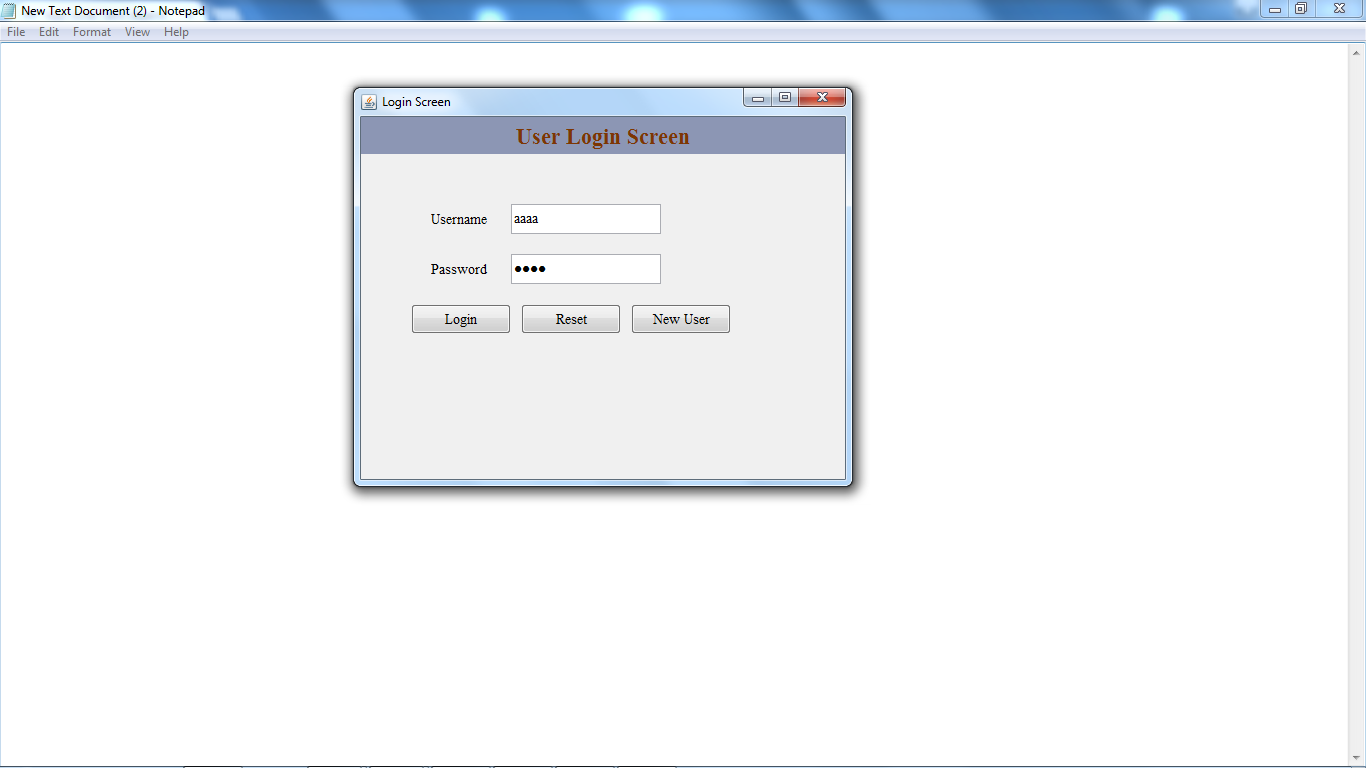
After successfully registering a data owner:



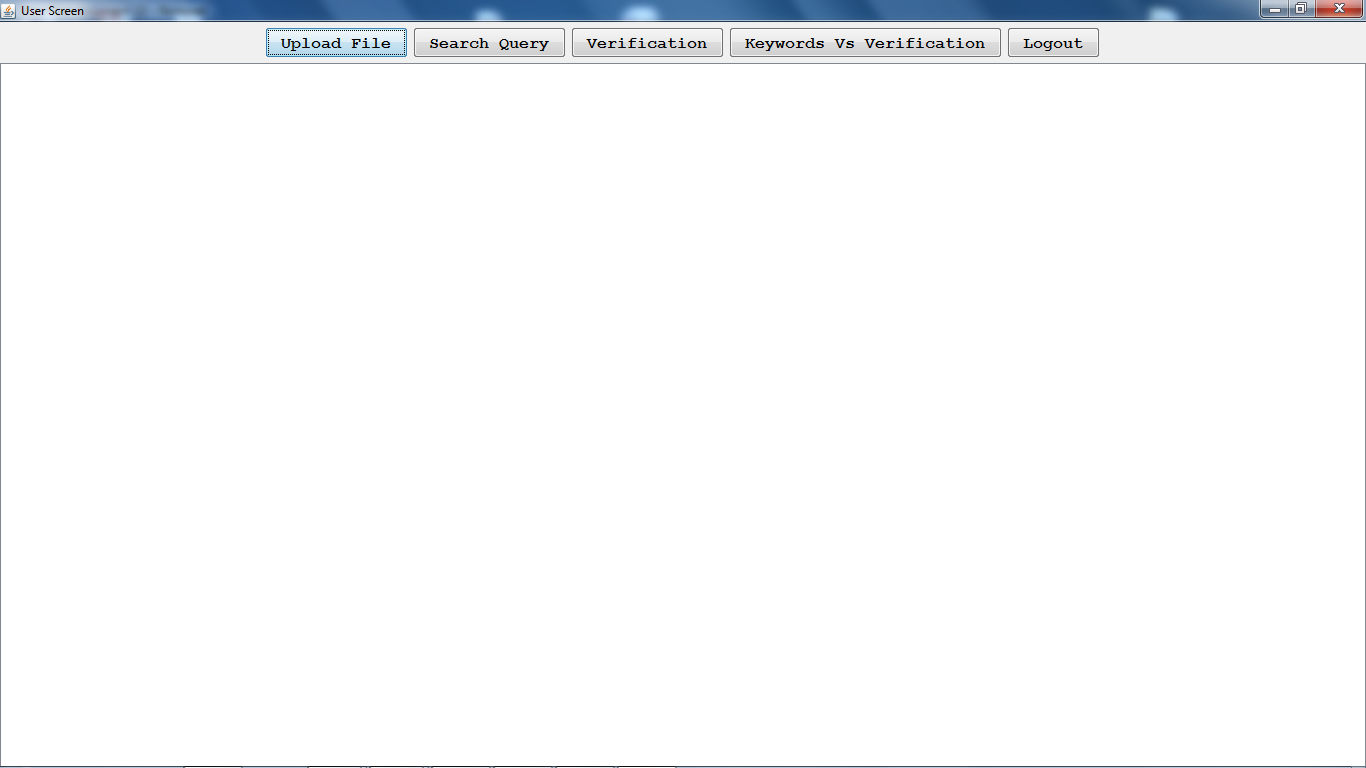
Registering another data owner:



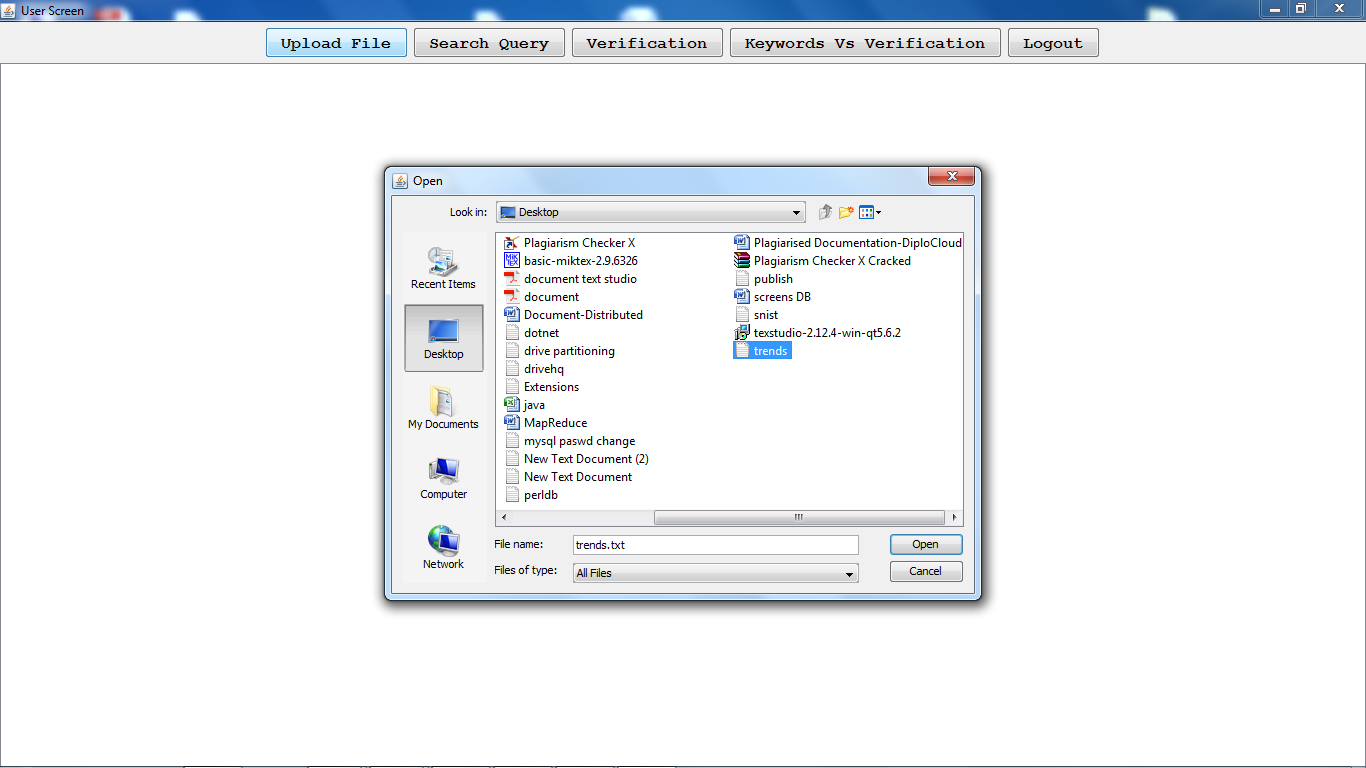
Login as a data owner:



Data owner home screen:

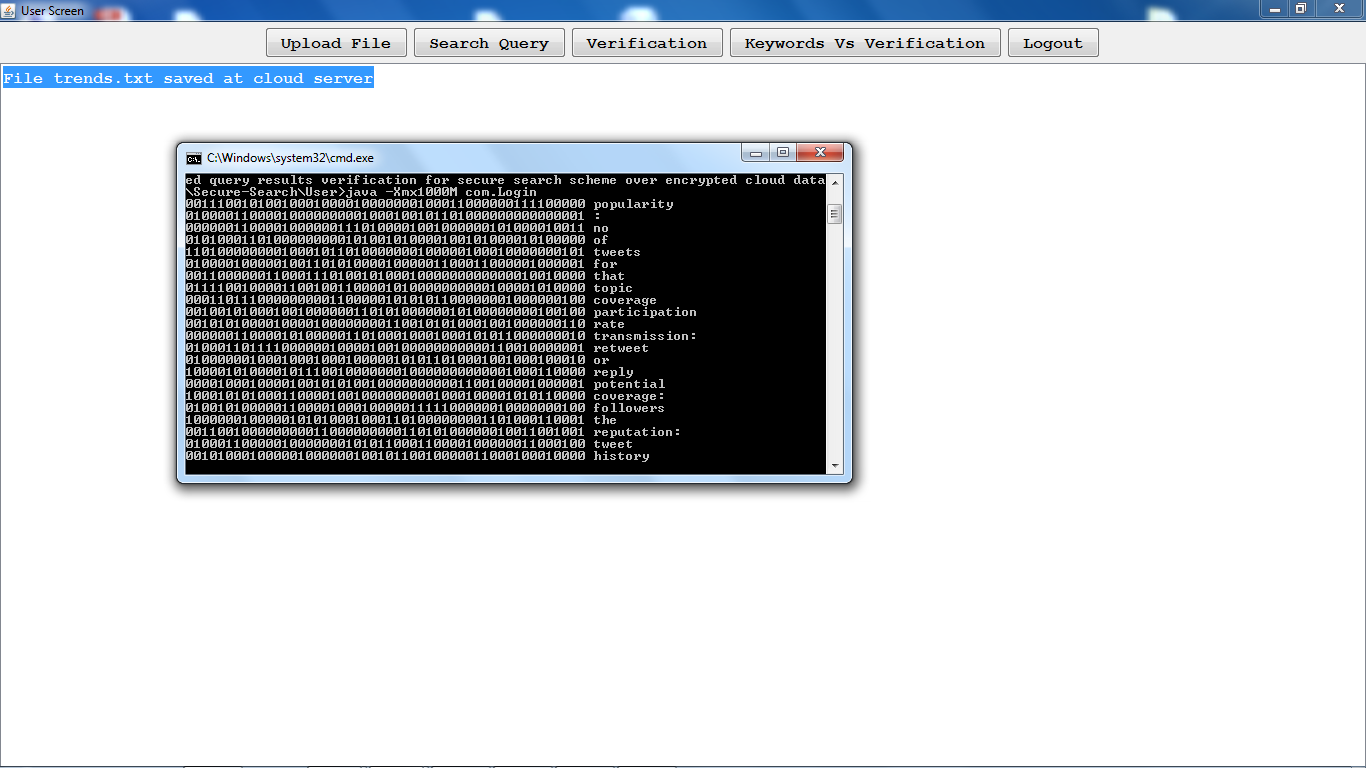


Upload a file onto cloud:

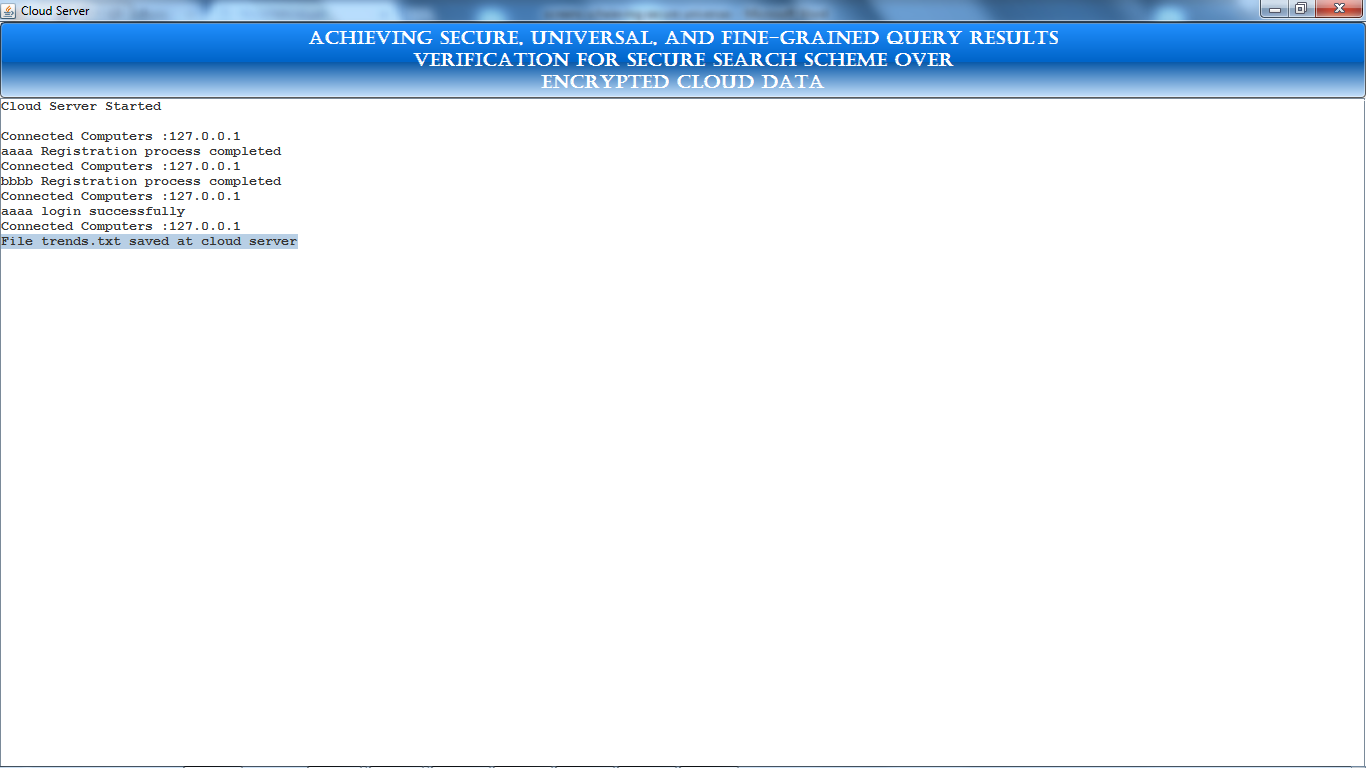


After uploading the file on to cloud

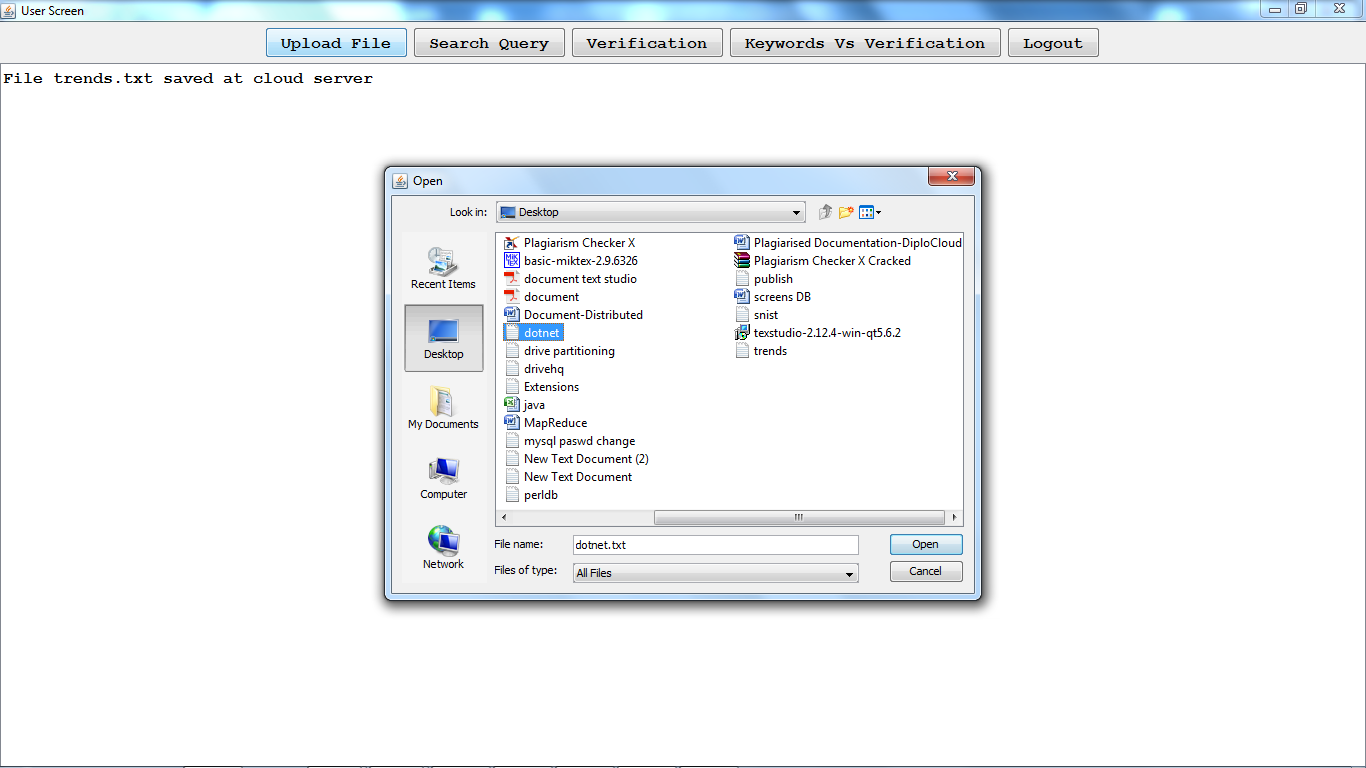
(the generated bloom filter..First for the given word get the bigrams, then generate hash for each bigram by using Paillier Encryption technique then generate some integer value for each hash code (here we are taking the array range up to 50) then generate bloom filter signature)



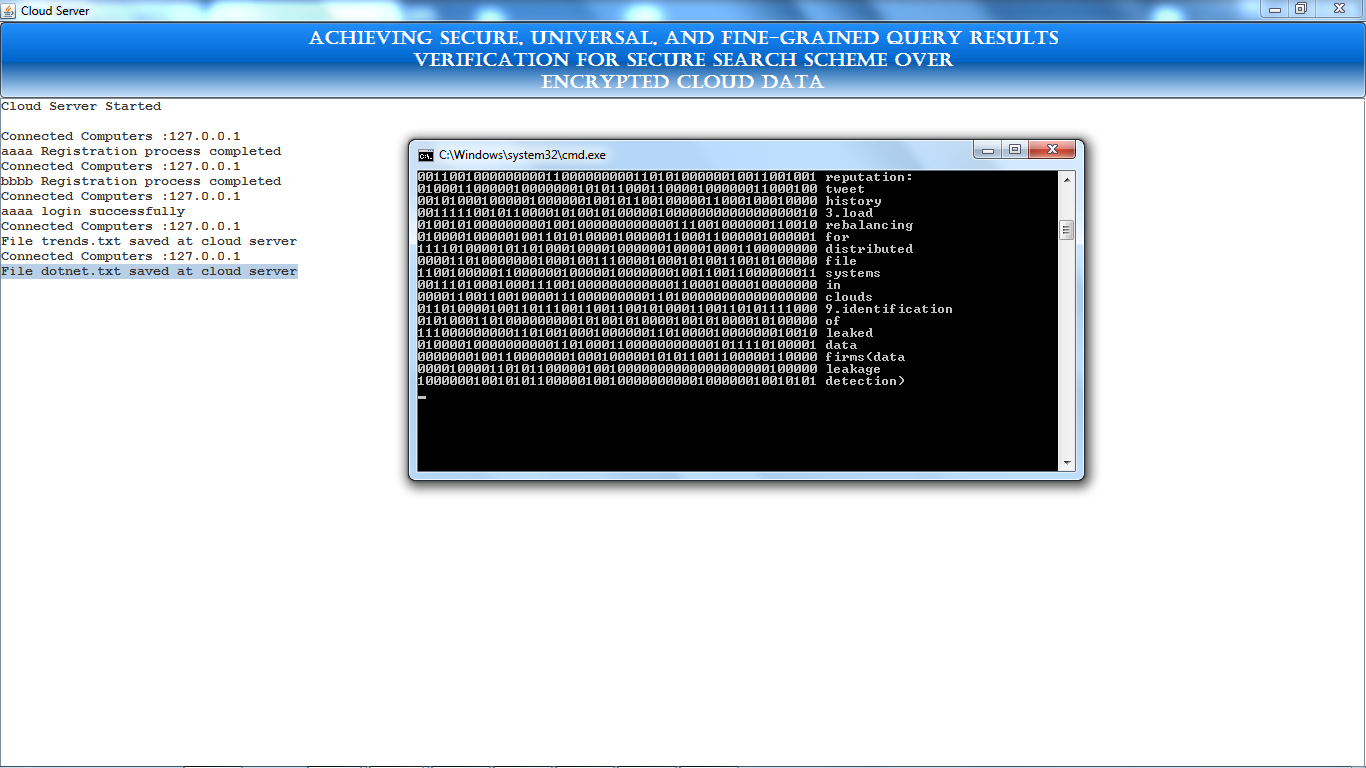
Cloud server after uploading the file: (the file will be saved at cloud in encrypted format by using AES algorithm)



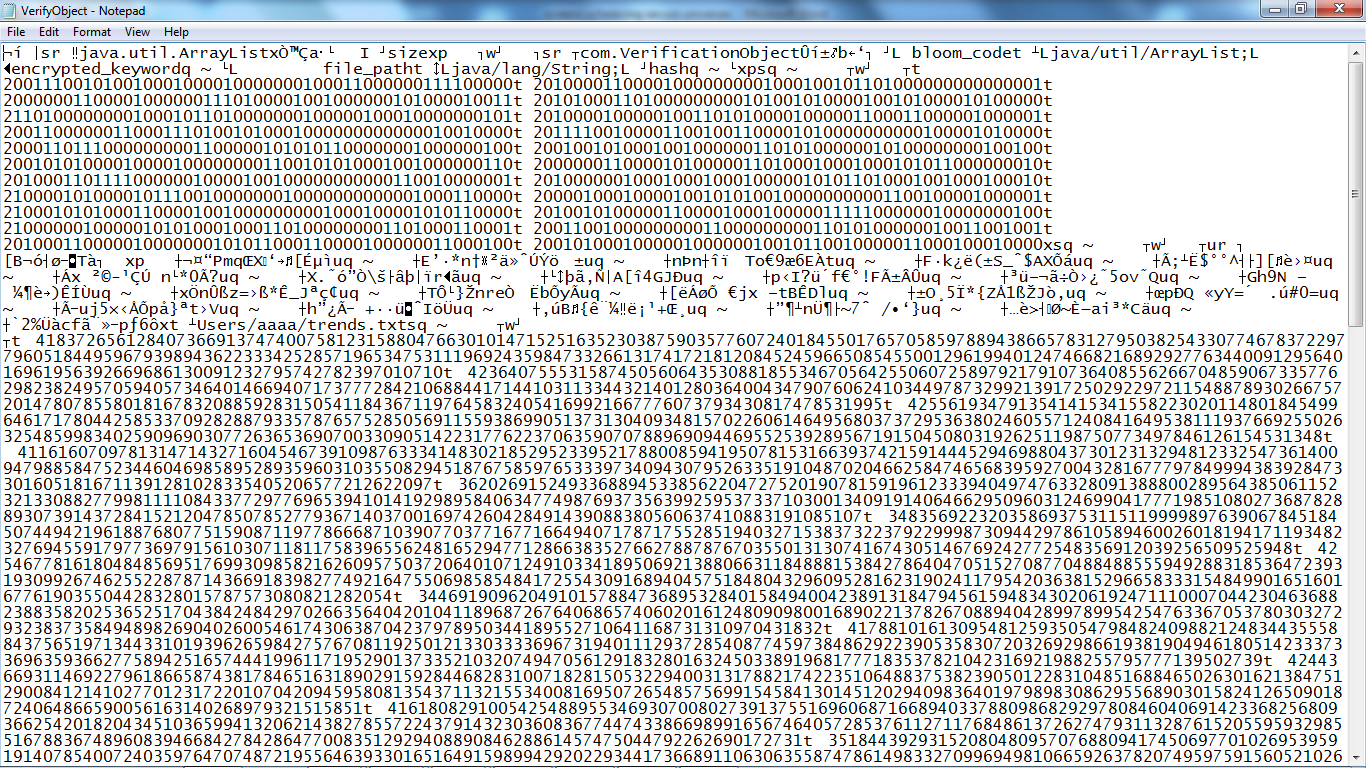
Uploading some other file onto cloud:



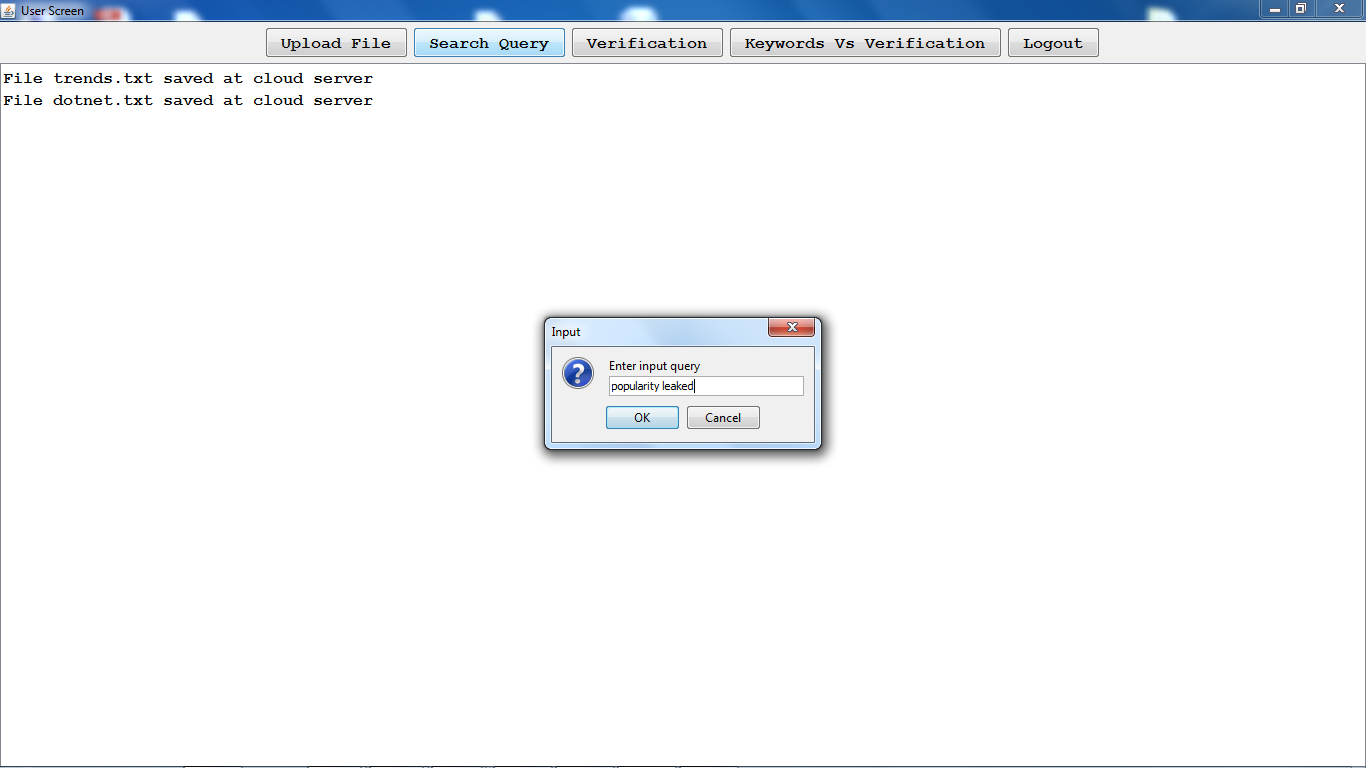
After successfully uploading:



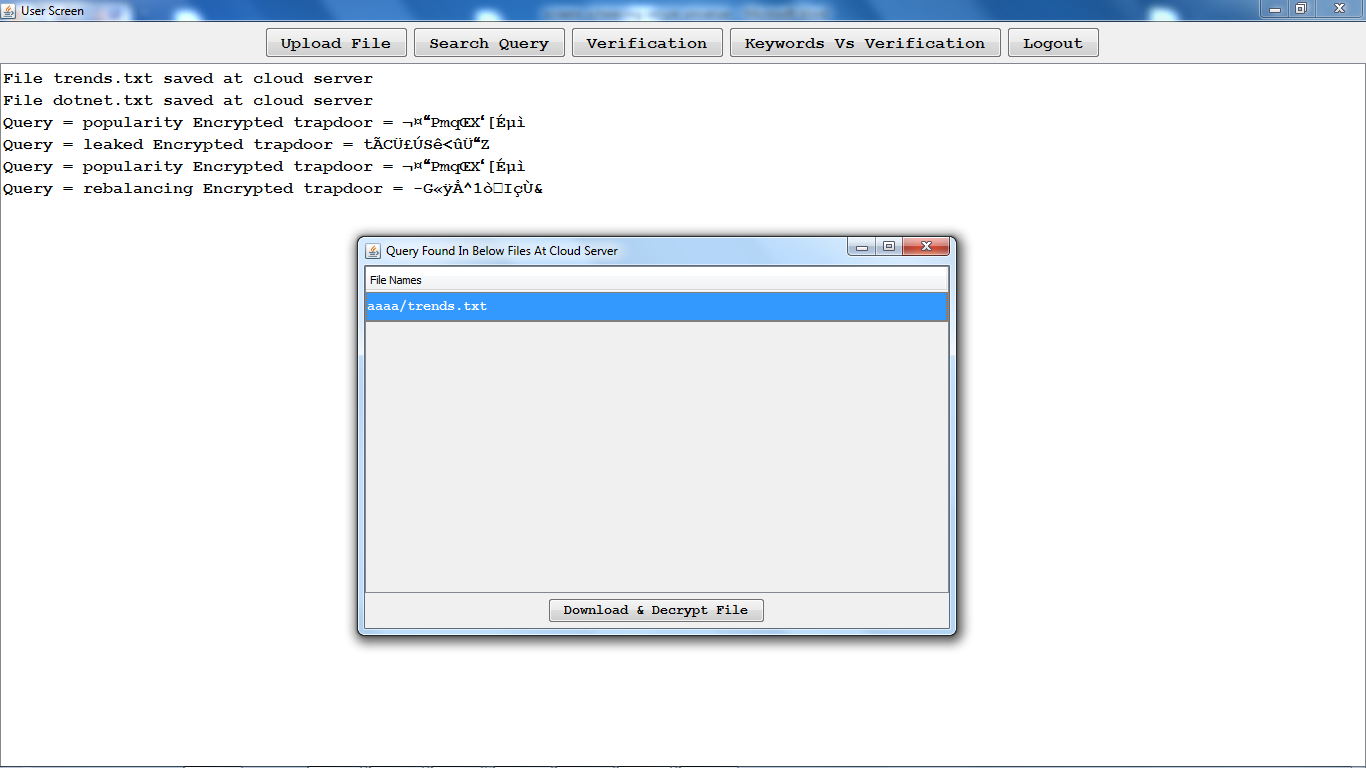
The verification object for the uploaded files will be generated & saved at server side:



Search query (enter the keywords from both files) (as per this project to reduce the resources, we can give the search results from few of the uploaded files if the client is satisfied with that then he can download that file otherwise he will keep on search until he will receive the required docs)

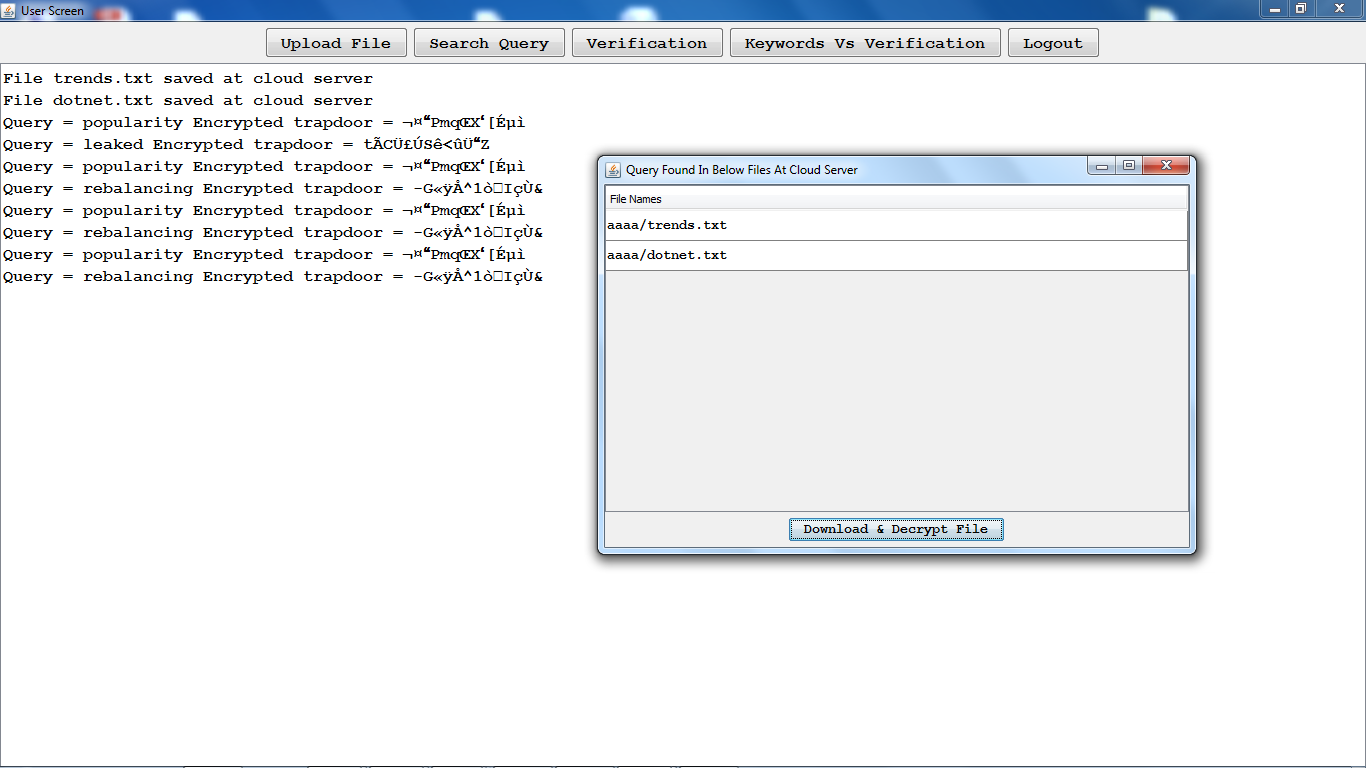


Then for the given query, trapdoor (hashcode) will be generated & searched from verification object for the given hashcode

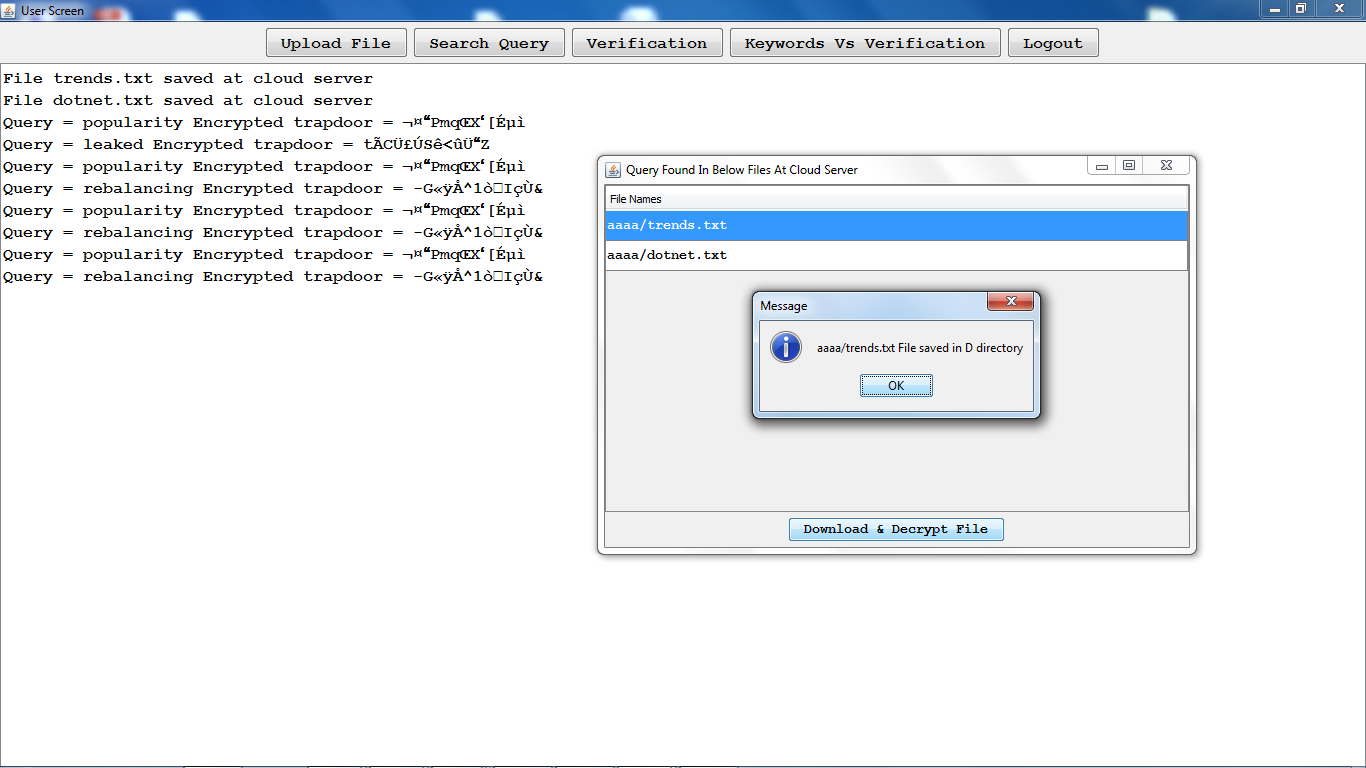


Here it shows the results from one file.

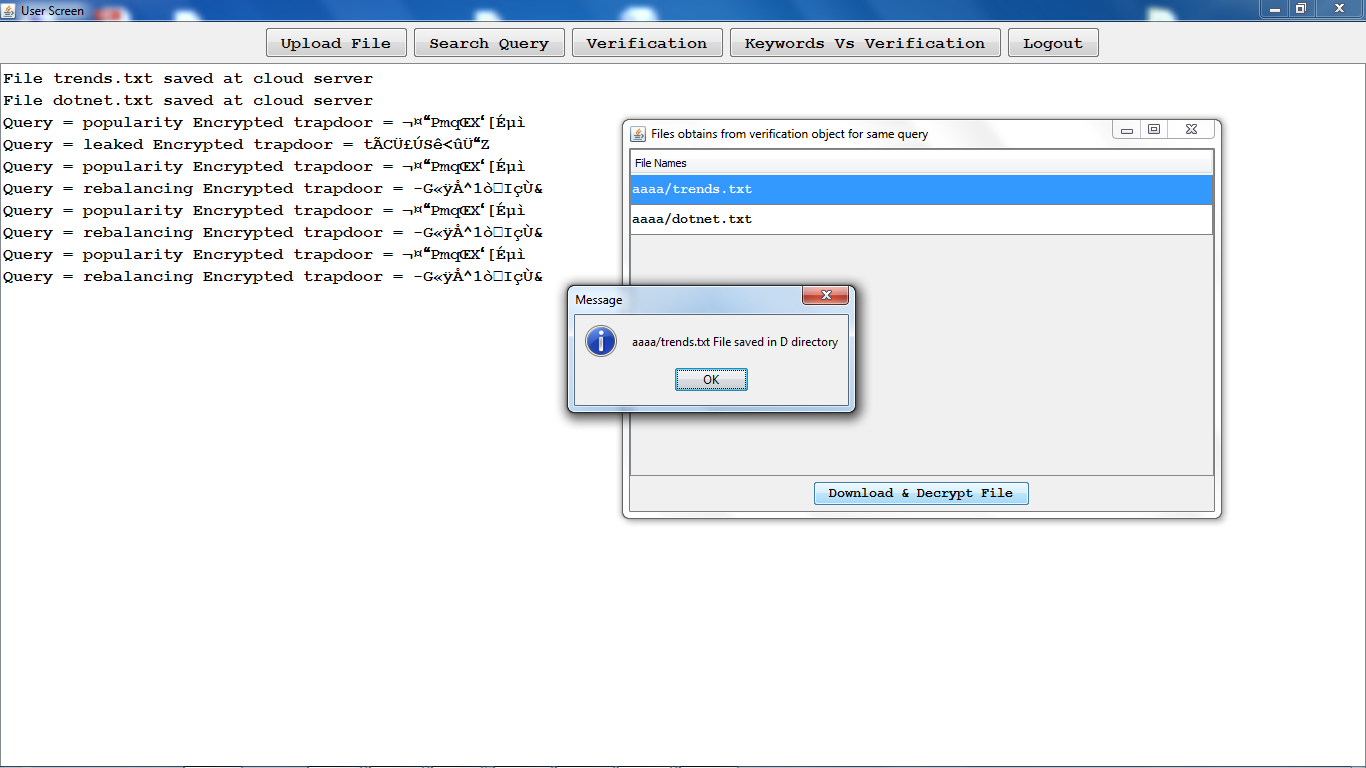
So if the user satisfied then he can download or else he will keep on search until he will get the desire output:



Select the required file & decrypt & download:

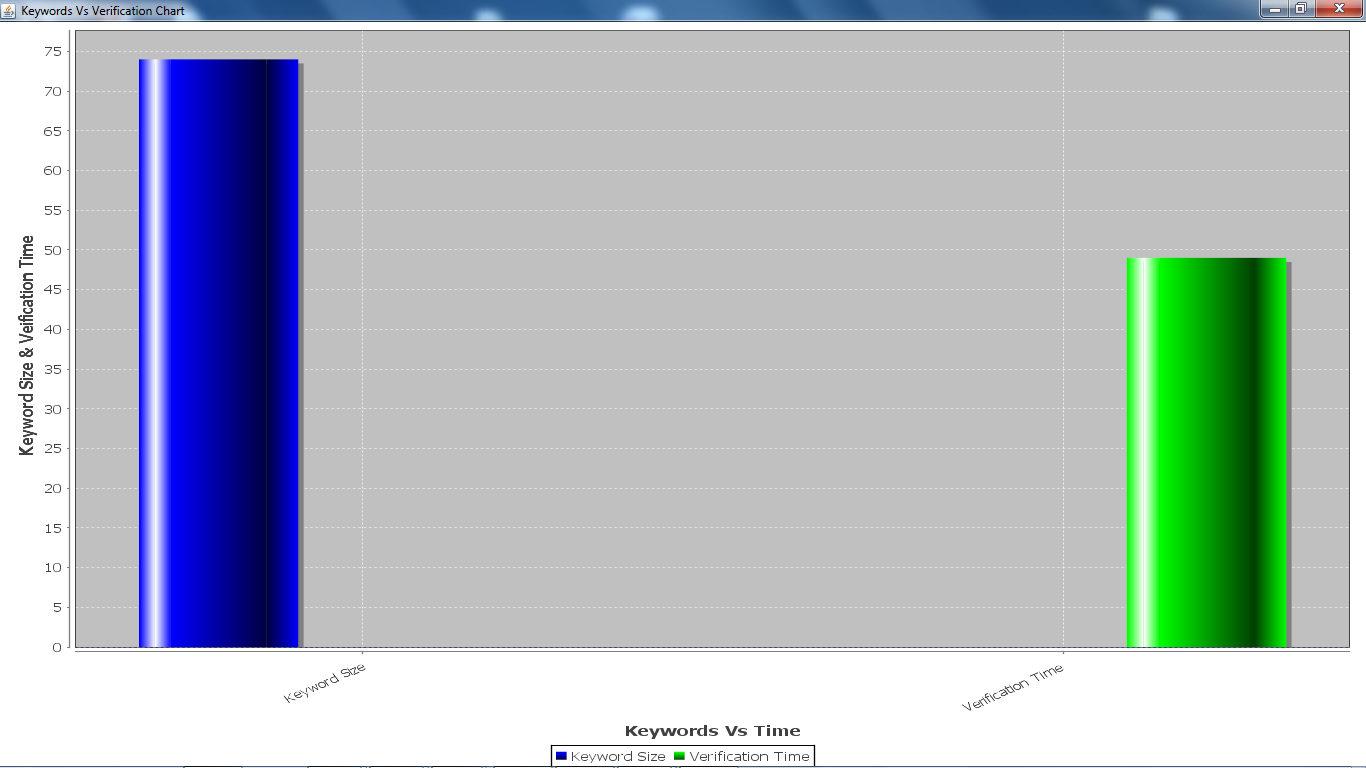


Verification:

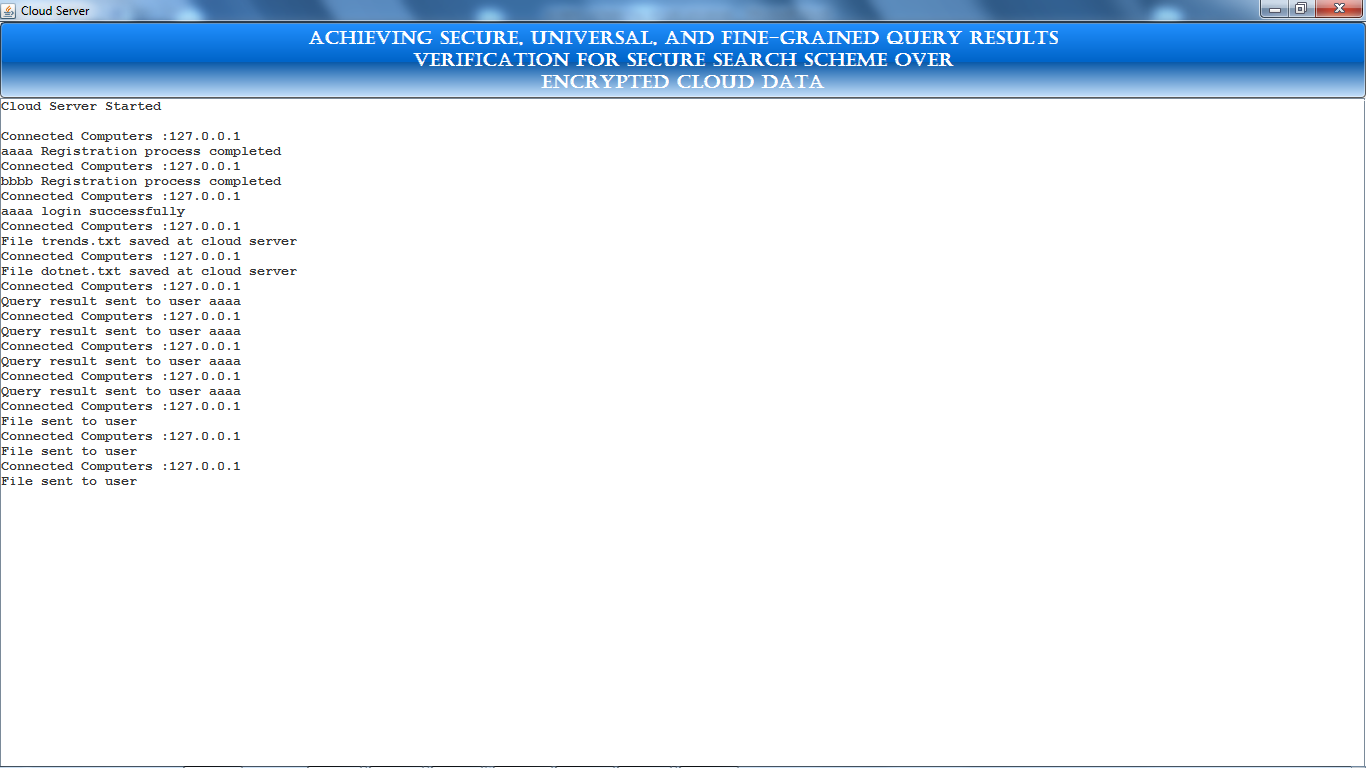


Keywords Vs verification chart:

(How many keywords are there in the uploaded files & how much time it has taken to verify)



The cloud server:



**8. CONCLUSION**

In this document, we propose a safe, effectively incorporated, & fine-grained question comes about check plot for secure inquiry over encoded cloud information. Not the same as past works, our plan can check the rightness of each encoded question result or further precisely discover what number of or which qualified information records are returned by the exploitative cloud server. A short mark procedure is intended to ensure the validness of confirmation question itself. Additionally, we plan a protected confirmation question ask for system, by which the cloud server knows nothing about which check protest is asked for by the information client & really returned by the cloud server. Execution & precision tests exhibit the legitimacy & effectiveness of our proposed conspires.

**REFERENCES**

1. K. Ren, C. Wang, & Q. Wang, “Security challenges for the public cloud,” IEEE Internet Computing, vol. 16, no. 1, pp. 69–73, 2012
2. S. Kamara & K. Lauter, “Cryptographic cloud storage,” in Springer RLCPS, January 2010
3. D. Song, D. Wagner, & A. Perrig, “Practical techniques for searches on encrypted data,” in IEEE Symposiumon Security & Privacy, vol. 8, 2000, pp. 44–55.
4. E.-J.Goh, “Secure indexes,” IACR ePrint Cryptography Archive, http://eprint.iacr.org/2003/216, Tech. Rep., 2003.
5. D. Boneh, G. D. Crescenzo, R. Ostrovsky, & G. Persiano, “Public-key encryption with keyword search,” in EUROCRYPR, 2004, pp. 506–522.
6. R. Curtmola, J. Garay, S. Kamara, & R. Ostrovsky, “Searchable symmetric encryption: improved deinitions & efficient constructions,” in ACM CCS, vol. 19, 2006, pp. 79–88.
7. M. Bellare, A. Boldyreva, & A. O’Neill, “Deterministic & efficiently searchable encryption,” in Springer CRYPTO, 2007.
8. K. Kurosawa & Y. Ohtaki, “Uc-secure searchable symmetric encryption,” Lecture Notes in Computer Science, vol. 7397, pp. 258– 274, 2012.
9. P. Xu, H. Jin, Q. Wu, & W. Wang, “Public-key encryption with fuzzy keyword search: A provably secure scheme under keyword guessing attack,” IEEE Transactions on Computers, vol. 62, no. 11, pp. 2266–2277, 2013.
10. S. Kamara & C. Papamanthou, “Parallel & dynamic searchable symmetric encryption,” in Financial Cryptography & Data Security. Springer Berlin Heidelberg, 2013, pp. 258–274