A Project Report

on

Searching Encrypted Data Without Decryption

Submitted for partial fulfillment of the requirements for the award of the degree

Of

BACHELOR OF ENGINEERING

IN

COMPUTER SCIENCE AND ENGINEERING

BY

Mr. Jithin Babu (2451-13-733-024)

Mr. Ch. Siddi Avinash (2451-13-733-033)

Mr. J. Daniel Kanakambaram (2451-13-733-036)

Under the guidance of

V. Sridhar

Assistant Professor

Department of CSE

M.V.S.R.E.C., Hyderabad.

Department of Computer Science and Engineering

M.V.S.R. ENGINEERING COLLEGE

(Affiliated to Osmania University & Recognized by AICTE)

Nadergul, Saroor Nagar Mandal, Hyderabad – 501 510

2016-17.

Department of Computer Science and Engineering

M.V.S.R. ENGINEERING COLLEGE

(Affiliated to Osmania University & Recognized by AICTE)

Nadergul, Saroor Nagar Mandal, Hyderabad – 501 510

CERTIFICATE

This is to certify that the project work entitled **"Searching Encrypting Data Without Decryption"** is a bonafide work carried out by **Mr. Jithin Babu (2451-13-733-24), Mr. Ch. Siddi Avinash (2451-13-733-033)**, **Mr. J. Daniel Kanakambaram (2451-13-733-036)** in partial fulfillment of the requirements for the award of degree of **BACHELOR OF ENGINEERING IN COMPUTER SCIENCE AND ENGINEERING** from M.V.S.R. Engineering College, affiliated to OSMANIA UNIVERSITY, Hyderabad, under our guidance and supervision.

The results embodied in this report have not been submitted to any other university or institute for the award of any degree or diploma.

Internal Guide	Head of the Department
V. Sridhar	**Dr. Akhil Khare**
Assistant Professor	Professor and Head
Department of CSE	Department of CSE
MVSREC, Hyderabad.	MVSREC, Hyderabad.

DECLARATION

This is to certify that the work reported in the present project entitled "**Searching Encrypted Data Without Decryption**" is a bonafide work done by us in the Department of Computer Science and Engineering, M.V.S.R. Engineering College, Osmania University. The reports are based on the project work done entirely by us and not copied from any other source.

The results embedded in this project report have not been submitted to any other University or Institute for the award of any degree or diploma to the best of our knowledge and belief.

Mr. Jithin Babu	Mr. Ch. Siddi Avinash	Mr. J Daniel Kanakambaram
(2451-13-733-024)	(2451-13-733-033)	(2451-13-733-036)

ACKNOWLEDGEMENT

We take this opportunity to express our profound and sincere gratitude to all those who helped us in carrying out this project successfully.

We would like to thank our Internal Project Guide **Mr. V. Sridhar**, Assistant Professor, Department of Computer Science and Engineering, M.V.S.R. Engineering College for his useful suggestions, guidance and encouragement.

At the very outset, we are thankful to our principal **Dr. V. Chandra Sekhar** and **Dr. Akhil Khare,** Professor and Head, Department of Computer Science and Engineering, M.V.S.R. Engineering College, Hyderabad for their consent to do the project work as a part of our B.E Degree (CSE). We thank them for their valuable suggestions and advice throughout our stay at the college during our project work.

We thank the teaching and non-teaching staff of CSE for extending their support.

Finally, we are thankful to our parents for their cooperation and support throughout all endeavors in our life.

Mr. Jithin Babu	(2451-13-733-024)
Mr. Ch. Siddi Avinash	(2451-13-733-033)
Mr. J Daniel Kanakambaram	(2451-13-733-036)

List of Figures

Fig 2.1 System Architecture 18

Fig 2.2 Threats 19

Fig 2.3 Onion Layers 21

Fig 4.1 Class Diagram....................35

Fig 4.2 Activity Diagram 1 36

Fig 4.3 Activity Diagram 2 37

Fig 4.4 Deployment Diagram 37

Fig 4.5 System Architecture 38

Fig 4.2.1 Compilation and Interpretation of Java 40

Fig 4.2.2 Types of Components....................41

Fig 4.2.3 Types of Containers 42

Fig 5.2.1 No of documents indexed 63

Fig 5.2.2 Indexing process 1 64

Fig 5.2.3 Indexing Process 2 64

Fig 5.2.4 Encrypting tf-idf values in documents 65

Fig 5.2.5 List of tables in CryptDB 65

Fig 5.2.6 Encrypted values in the database in table ‘b’ 66

Fig 5.2.7 Encrypted values in the database in table ‘f’ 66

Fig 5.2.8 Inserting into terms and corresponding tf-idf values into CryptDB....................67

Fig 5.2.9 Reading the query from the user....................67

Fig 5.2.10 Displaying the documents after ranking them....................68

ABSTRACT

Nowadays every user who uses Internet wants to search for anything and everything using Search Engines. This is the need for everyone. To engineer a search engine is a challenging task. Search engines index tens to hundreds of millions of files involving a comparable number of distinct terms. They answer tens of millions of queries every day.

Due to rapid advance in technology and need for information security, creating a web search engine today is very different from three years ago. The overall goal of this project is to develop a scalable, high performance search engine which searches the encrypted data without decryption. The main focus is on the algorithmic challenges and encryption while supporting fast searches on it.

To develop this project, we applied ranking algorithm to give better results to the user. We also used other algorithms to encrypt the data stored. To ease for searching of various information over the data by giving search keywords requires a software. The search engine software ensures the end user to get the information by accessing the data specified in the database.

Table of Contents

Title i

Certificate ii

Declaration iii

Acknowledgement iv

List of figures v

Abstract vi

Table of Contents vii

CHAPTER 1 1-4

INTRODUCTION

1.1 Introduction 1

1.2 Motivation 3

1.3 Objectives 4

CHAPTER 2 5-34

LITERATURE SURVEY

2.1 Information Retrieval System: the Vector Model 5

2.1.1 Term Dependencies in Document Indexes 7

2.1.2. Document Indexing Based on Term Contexts 10

2.1.2.1 Generating Term Context Vectors 11

2.1.2.2 Generating Document Context Vectors 12

2.1.3 Document Retrieval 13

2.1.3.1 Term Weights 15

2.1.3.2 Deviation measures of terms across document vectors 16

2.2 CryptDB 17

2.2.1 System Architecture 18

2.2.2 Onion layers 21

2.2.3 Encryption types 22

2.2.3.1 Random (RND) 22

2.2.3.2 Homomorphic encryption (HOM) 22

2.2.3.3 Word search (SEARCH) 23

2.2.3.4 Deterministic (DET) 23

2.2.3.5 Order-preserving encryption (OPE) 24

2.2.3.6 Join (JOIN,OPE-JOIN) 24

2.2.4 CryptDB security related papers 25

2.2.5 Discussion and Limitations 26

2.2.6 Implementation 27

2.2.7 Guidelines 28

2.2.8 Installation and Configuration of CryptDB 31

2.2.9 Conclusion 32

CHAPTER 3 33-34

PROBLEM DESCRIPTION

3.1 Problem Description 33

3.2 Minimum Hardware Requirements 34

3.3 Software Requirements 34

CHAPTER 4 35-60

SYSTEM DESIGN

4.1 UML Diagrams 35

4.2 Introduction of Technologies used........ 38

4.3 MySQL 53

4.4 Apache Lucene........ 57

4.4.1 Scalable, High-Performance Indexing 57

4.4.2 Powerful ,Accurate and Efficient Search Algorithms........ 57

4.4.3 Cross-Platform Solution........ 57

4.5 Implementation 58

4.5.1 DocIndexer.java 58

4.5.2 CryptCon.java 59

4.5.3 CosineSimilarity.java........ 59

4.5.4 Searcher.java........ 60

CHAPTER 5 61-68

TESTING AND RESULTS

5.1 Testing 61

5.1.1 System Testing........ 62

5.1.2 Module Testing 62

5.1.3 Integration Testing 62

5.1.4 Acceptance Testing........ 62

5.2 Screenshots 63

CHAPTER 6 69

CONCLUSION

6.1 Future Enhancements........ 69

CHAPTER 7 70-72

REFERENCES

CHAPTER 8 73-86

APPENDICES

8.1 DocIndexer.java 73

8.2 Cryptcon.java 79

8.3 Searcher.java 81

8.4 CosineSimilarity.java 85

1. INTRODUCTION

1.1 Introduction

Online applications are vulnerable to theft of sensitive information because adversaries can exploit software bugs to gain access to private data, and because curious or malicious administrators may capture and leak data. CryptDB is a system that provides practical and provable confidentiality in the face of these attacks for applications backed by SQL databases. It works by executing SQL queries over encrypted data using a collection of efficient SQL-aware encryption schemes. CryptDB can also chain encryption keys to user passwords, so that a data item can be decrypted only by using the password of one of the users with access to that data. As a result, a database administrator never gets access to decrypted data, and even if all servers are compromised, an adversary cannot decrypt the data of any user who is not logged in. An analysis of a trace of 126 million SQL queries from a production MySQL server shows that CryptDB can support operations over encrypted data for 99.5% of the 128,840 columns seen in the trace. Our evaluation shows that CryptDB has low overhead, reducing throughput by 14.5% for phpBB, a web forum application, and by 26% for queries from TPC-C, compared to unmodified MySQL. Chaining encryption keys to user passwords requires 11-13 unique schema annotations to secure more than 20 sensitive fields and 2-7 lines of source code changes for three multi-user web applications.

Theft of private information is a significant problem, particularly for online applications. An adversary can exploit software vulnerabilities to gain unauthorized access to servers; curious or malicious administrators at a hosting or application provider can snoop on private data; and attackers with physical access to servers can access all data on disk and in memory. One approach to reduce the damage caused by server compromises is to encrypt sensitive data, and run all computations (application logic) on clients. Unfortunately, several important applications do not lend themselves to this approach, including database-backed web sites that process queries to generate data for the user, and applications Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Even when this approach is tenable, converting an existing server-side application to this form can be difficult. Another approach would be to consider theoretical solutions such as fully homomorphic encryption, which allows servers to compute arbitrary functions over encrypted data, while only clients see decrypted data. However, fully homomorphic encryption schemes are still prohibitively expensive by orders of magnitude.

The cleverest part of CryptDB, however, is that it's able to switch between crypto systems on the fly depending on the operation. The data in the database is encrypted in multiple layers of different encryption, what the researchers call an "onion" of encryption. Every layer allows different kinds of computation and has a different key. The most secure schemes are used on the outside of that onion and the least secure are used on the inside. CryptDB manages to perform all its functions of a database without ever removing that last layer of the onion, so that the data always remains secure.

CryptDB has its limits, the MIT researchers warn--no square roots, for one example. And while the data is never completely decrypted, it does "leak" information about the underlying data when enough outer layers of encryption are removed, revealing attributes like which data points are equal to each other. But in their paper they sampled operations from several real databases like one used by a Web forum and another by a grade-calculating application, and found that their encrypted system would allow the same calculations as an unencrypted database in 99.5% of those operations, and that data the researchers deemed "sensitive" is never leaked in those test cases.

1.2 Motivation

Leakage of confidential data plagues many computing systems today. For example, last year marks a peak in data breaches: about 740 million records were exposed, the largest number so far. Most applications store sensitive data at servers, so preventing data leakage from servers is a crucial task towards protecting data confidentiality.

One potential approach to protecting confidentiality is to encrypt the data stored at the server and never send the decryption key to the server. This approach is promising because an attacker at the server can access only encrypted data and thus does not see the data content. However, this approach cannot be applied for most systems, such as databases, web applications, mobile applications, machine learning tools, and others. The reason is that these systems need to compute on the data and simply encrypting the data no longer allows such computation. Hence, existing systems have a different strategy.

In fact, almost all systems deployed today follow the same strategy: try to prevent attackers from breaking into servers. The reasoning behind this strategy is that, if attackers cannot break into servers, they cannot access the data at these servers. This strategy has been implemented using a multitude of approaches: checks at the operating system level, language-based enforcement of a security policy, static or dynamic analysis of application code, checks at the network level, trusted hardware, and others.

Nevertheless, data still leaks with these approaches. It turns out that attackers eventually break in. To understand why it is hard to prevent attackers from getting access to server data, we now provide examples of common attackers and the reason they manage to subvert existing systems. First, hackers notoriously break into systems by exploiting software bugs and gain access to sensitive data. They succeed because software is complex and hard to make bug free. In some cases, the hackers even manage to get administrative access to the servers, thus gaining access to all the data is stored there. The second threat appears in the context of cloud computing. More and more companies are outsourcing their data to external clouds, so their sensitive data becomes readily available to curious cloud administrators. Most existing protection mechanisms do not prevent such attackers because some cloud employees have full access to the servers and can simply bypass these mechanisms. Finally, according to recent news, the government has been accessing a lot of private data even without subpoena. In some cases, the government leverages physical access to the servers, making it hard to protect the data stored on these. Hence, despite existing protection systems, attackers eventually break in or bypass the protection, thus getting access to the data stored on servers.

1.3 Objectives

- To provide secure communication link for searching documents.
- Users should be able to provide the query.
- System should process the query and return the relevant documents after ranking them.

2. LITERATURE SURVEY

2.1. Information Retrieval System: The Vector Model

The vector space model is one of the classical and widely applied retrieval models to evaluate relevance of web page. The retrieval operation consists of computing the cosine similarity function between a given query vector and the set of documents vector and then ranking documents accordingly.

In the vector space model for information retrieval, term vectors are pair-wise orthogonal, that is, terms are assumed to be independent. It is well known that this assumption is too restrictive. In this article, we present our work on an indexing and retrieval method that, based on the vector space model, incorporates term dependencies and thus obtains semantically richer representations of documents. First, we generate term context vectors based on the co-occurrence of terms in the same documents. These vectors are used to calculate context vectors for documents. We present different techniques for estimating the dependencies among terms. We also define term weights that can be employed in the model. Experimental results on four text collections (MED, CRANFIELD, CISI and CACM) show that the incorporation of term dependencies in the retrieval process performs statistically significantly better than the classical vector space model with IDF weights. We also show that the degree of semantic matching versus direct word matching that performs best varies on the four collections. We conclude that the model performs well for certain types of queries and, generally, for information tasks with high recall requirements. Therefore, we propose the use of the context vector model in combination with other, direct word-matching methods.

In traditional keyword-based IR systems, the documents of a collection are represented by a set of keywords that describe their content. These representations are matched to the words describing the user's information need. Such systems have two fundamental problems:

i. the queries have to be specified using the same set of keywords that has been used during document indexing, and

ii. the keywords are usually assigned manually to the documents.

More modern IR systems find a way around the problem of a restricted search vocabulary and the subjectiveness of the indexing process by automating this process. There, the representation of a document is based on the words that occur in the text or in some surrogate (e.g., abstract). The set of keywords, or, as it is commonly known in full text indexing methods, the set of index terms, includes all the words occurring in the collection. This set is usually confined to only the significant words by eliminating common functional words (also called stop words). However, this indexing approach has brought new problems. The first problem is known in the IR community as the vocabulary or word-matching problem. It refers to the fact that different documents describing the same subject may use different words. In such cases, a simple word-matching approach will probably miss some relevant documents, just because they do not contain the same terms as used in the query. The second problem is the growing need for mechanisms that rank documents in order of relevance, since many more documents are likely to match the words occurring in a query.

Probably the best known model in IR is the Vector Space Model (VSM) (Salton, 1989; Salton & McGill, 1983). It implements full-text automatic indexing and relevance ranking. In VSM, documents and queries are modeled as elements of a vector space. This vector space is generated by a set of basis vectors that correspond to the index terms. Each document can be represented as a linear combination of these term vectors. The indexing process thus consists of calculating the document vectors. During the retrieval process, a query is also put through the indexing process and a query vector is obtained. This query vector is then matched against each document vector and a retrieval status value (e.g. cosine coefficient) is calculated that measures the similarity or aboutness of the document to the query. As a result the retrieval process returns a list of all documents ordered by the calculated retrieval status values (relevance to the query).

The effectiveness of retrieval models is usually measured in terms of precision and recall. Precision is the ratio of the number of relevant retrieved documents to the total number of retrieved documents. Recall is the proportion of relevant documents retrieved. In Boolean IR systems, where documents are directly retrieved if they satisfy the logical constraints of the query, precision and recall can be calculated directly. In ranking models such as VSM, the calculation of these values is not straightforward since a list of all documents is retrieved. There, common evaluation methods are

i. precision at different recall levels (e.g. after 10%, 20%, …, 100% of the relevant documents have been retrieved),
ii. precision after a fixed number of retrieved documents, and
iii. average precision after all relevant documents, where the precision values are calculated up to each relevant document and these values are averaged.

One assumption of the classical vector space model is that term vectors are pair-wise orthogonal. This means that when calculating the relevance of a document to a query, only those terms are considered that occur in both, the document and the query. Using the cosine coefficient, for example, the retrieval status value for a document/query pair is only determined by the terms the query and the document have in common, but not by query terms that are semantically similar to document terms or vice versa. In this sense the model assumes that terms are independent of each other; there exists no relationship among different terms. It is well known that this assumption is too restrictive, since words are not actually independent. They do have relationships, because they represent concepts or objects that may be similar or related by many different types of semantic associations. Although the vector space model does not include term dependencies and does not solve the word-matching problem, it has turned out to be very effective, and many other more complex models have not achieved the expected substantial improvement in retrieval performance.

2.1.1. Term Dependencies in Document Indexes

One of the fundamental problems in information retrieval is the vocabulary or word matching problem. It actually refers to two different things: i) the same objects may be expressed in different ways so documents about the same issue may use different words, and ii) the existence of words that have different meanings. The first is called synonymy and the second polysemy. In ranking IR systems, the prevalence of synonyms tends to decrease precision at higher recall levels (e.g. the last relevant documents found are further down in the list), since not all of the relevant documents may match the terms in a query. On the other hand, the existence of polysemic terms is related to low precision at low recall levels, that is, there will be less relevant documents at the beginning of the list. If a term with an unclear meaning is used in a query then many irrelevant documents, which contain the term but not with the intended meaning, are likely to be retrieved earlier. Polysemy and synonymy actually describe just the extreme cases and there is a broad spectrum of relationships between words and their meanings between the two.

There have been many attempts to solve the vocabulary problem. It has been argued that the use of term dependencies can help to achieve this aim (Bollmann-Sdorra & Raghavan, 1998; Raghavan & Wong, 1986) and almost all methods use such term dependencies in one way or another. Most of these methods get term relationships from co-occurrence data, that is, from the frequency terms co-occur in the same documents. Schütze (1992) represents the semantics of words and contexts in a text as vectors in a vector space where the dimensions correspond to words. These vectors are called context vectors. A context vector for an entity is obtained from the words occurring close to that entity in a text. Therefore, the vectors represent the context of a single occurrence of a word. Because of the high dimensionality of the vectors, dimensionality reduction is carried out by means of singular value decomposition. The vectors are later applied to word sense disambiguation and thesaurus generation tasks. Even though the use of the methods is only reported for these tasks, they could be used directly in IR systems as well.

Many of the methods approach the vocabulary problem by means of query expansion. Query expansion techniques either consider the information search as a process and try to refine a user's query in each retrieval step (Chen et al., 1997; Chen et al. 1995) or they use inter-term relationships to expand a query when it is received. The information on term dependencies may come from manually or automatically generated thesauruses, as in (Gotlieb & Kumar, 1968; Chen et al., 1997; Chen et al. 1995; Jing & Tzoukermann, 1999), or may be obtained by means of relevance feedback. In relevance feedback, the system analyses the documents a user judged relevant at a previous stage and uses this information for query refinement. It has been shown that automatic query expansion using relevance feedback can add useful words to a query and can improve retrieval performance (Salton & Buckley, 1990). However, such techniques require the collaboration of the user who has to judge the documents supplied and such judgments are often not provided. Ad hoc or blind feedback can solve this problem (Buckley et al., 1995; Mitra, Singhal, & Buckley, 1998). Blind feedback assumes that the top retrieved documents of a retrieval process are likely to be relevant and that shifting the query towards these documents will improve the retrieval results. In a blind feedback approach, documents are first retrieved for the original query. Then the top n documents are selected and used in a relevance feedback process to create an expanded query. Usually those terms that are good representatives of the selected documents are added to the query. Afterwards, the final document ranking is obtained by repeating the retrieval process with the new, expanded query. Besides the effect of shifting the query towards a region of possibly relevant documents in the vector space, blind feedback expands the query by adding more terms to it. It is likely that new terms are added, which are semantically related to the original query terms. Thus, probably more relevant documents will match the terms in the new query. On the other hand, non-relevant documents that matched some terms in the original query, but use these terms in a different meaning, may now get a lower retrieval status value since the new query terms may not be present. Blind feedback can improve the retrieval performance considerably if the selected documents are actually relevant. If this is not the case, however, the performance may also decrease.

Another class of methods approach the vocabulary problem by generating representations of documents and queries that are semantically richer than just vectors based on the occurrence frequency of terms. They use the inherent semantic structure that exists in the association of terms with documents in the indexing process. In Latent Semantic Indexing (LSI) (Deerwester et al., 1990), singular value decomposition is used to decompose the original term/document matrix into its orthogonal factors. Of those, only the n highest factors are kept and all others are set to 0. The chosen factors can approximate the original matrix by linear combination. Thus, smaller and less important influences are eliminated from the document index vectors, and terms that did not actually appear in a document may now be represented in its vector. LSI can be seen as a space transformation approach, where the original vector space is transformed to some factor space. The Probabilistic Latent Semantic Indexing (PLSI) by Hofmann (1999) has a similar approach. Also, in PLSI the documents are mapped to a reduced vector space, the latent semantic space. As opposed to LSI, the model has a solid statistical foundation. It is based on a latent variable model for general co-occurrence data, which associates an unobserved latent class variable with each observation. The number of latent factors will be much smaller than the number of words and the factors act as prediction variables for words. The factors are obtained using a generalization of the Expectation Maximization algorithm. A transformation of the vector space is also the basis of the Generalized Vector Space Model (GVSM) (Wong et al., 1987). Wong et al. argue that the orthogonality assumption in VSM is too restrictive and, therefore, another representation has to be found. Analyzing term correlations obtained from co-occurrence data, they define a new set of orthogonal basis vectors that spans a (transformed) vector space. This space is then used to represent document and query vectors more semantically, since term dependencies are implicitly included. The results they report with their model show a clear improvement over the classical vector space approach.

2.1.2. Document Indexing Based on Term Contexts

The basic algorithm that we use consists of three steps: i) compute term/document matrix, ii) generate term context vectors, and iii) transform document vectors based on occurrence frequencies to document context vectors. Normally, the starting point of the indexing process in a vector space based method is a term/document matrix as shown below:

$$\begin{array}{c|cccc}
 & d_1 & d_2 & \cdots & d_m \\
\hline
t_1 & w_{11} & w_{21} & \cdots & w_{m1} \\
t_2 & w_{12} & w_{22} & \cdots & w_{m2} \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
t_n & w_{1n} & w_{2n} & \cdots & w_{mn}
\end{array}$$

m is the number of documents in the collection and *n* the number of index terms. Each element w_{ij} in this matrix corresponds to the occurrence frequency of term t_j in document d_i. The vectors $d_i = (w_{i1}, w_{i2}, \ldots , w_{in})_T$ are used as the initial document vectors in most information retrieval models that are based on the *bag of words* approach. In the classical

VSM, these vectors represent the documents of a collection. They are usually only modified for normalization purposes and through the introduction of term weights to improve retrieval performance (e.g., IDF weights). However, in our approach we use the initial occurrence frequency vectors as the starting point for calculating document context vectors.

2.1.2.1 Generating Term Context Vectors

Term context vectors can be seen as a semantic description of terms, which reflect the influences of terms in the conceptual descriptions of other terms. The set of term context vectors can be represented by an *n*×*n* matrix T as follows:

$$T = \begin{pmatrix} c_{11} & c_{21} & \cdots & c_{n1} \\ c_{12} & c_{22} & \cdots & c_{n2} \\ \vdots & \vdots & \ddots & \vdots \\ c_{1n} & c_{2n} & \cdots & c_{nn} \end{pmatrix}$$

where the i-th column represents the term context vector $t_i=(c_{i1}, c_{i2}, \ldots , c_{in})^T$ for the index term t_i in the *n* dimensional term space. Each c_{ik} represents the influence of term t_k on term t_i. The use of term context vectors allows us two different interpretations of index terms: i) a syntactic interpretation of a term as the word it stands for, and ii) as a semantic description of a concept underlying the word the term stands for. Term dependencies can be integrated into the indexing model by means of the second interpretation, that is, by describing each index term by a set of concepts, where each concept has a directly corresponding index term. In the following, we discuss how the elements of matrix T can be obtained. To estimate the influence of term t_k on the semantic description of t_j we use the co-occurrence frequency of both terms, that is, the frequency with which t_k and t_j co-occur in the same textual units across the whole collection.

We define two different co-occurrence frequency measures:

$$c_{ij} = \begin{cases} 1 & \text{, if } i = j \\ \dfrac{\sum_{k=1}^{m} w_{ki} w_{kj}}{\sum_{k=1}^{m} \left(w_{ki} \sum_{a=1, a \neq i}^{n} w_{ka} \right)} & \text{, otherwise} \end{cases}$$

$$c_{ij} = \begin{cases} 1 & \text{, if } i = j \\ \dfrac{\sum_{\substack{k=1, \\ \text{where } w_{kj} \neq 0}}^{m} w_{ki}}{\sum_{k=1}^{m} w_{ki}} & \text{, otherwise} \end{cases}$$

With both definitions, it is assured that for all $i \leq n$ and all $k \leq n$: $c_{ik} \leq 1$. Furthermore, for all diagonal elements c_{ii} it holds $c_{ii}=1$. These properties are quite important from an interpretative perspective. They imply that the influence of a term in its context vector is bigger than the influence of any other term on that term. As we see later, however, it may be advantageous to set these influences to 0. Then, only its relations to all other terms describe the semantic meaning of a term.

2.1.2.2. Generating Document Context Vectors

Once the term correlation matrix T has been generated, we transform each initial document vector into a context vector $\overrightarrow{d'}_i = (c_{i1}, c_{i2}, \dots\dots\dots, c_{in})^T$. This is done with the following equation:

$$\vec{d}'_i = \frac{\sum_{j=1}^{n} w_{ij} \dfrac{\vec{t}_j}{|\vec{t}_j|}}{\sum_{j=1}^{n} w_{ij}}$$

where $\vec{t}_i$ is the context vector of term t_j and $|\vec{t}_j|$ is the length of vector $\vec{t}_j$ (we use the Euclidean vector norm to define the length of a vector). The division of the elements in term context vectors by the length of the vector is just a normalization step. However, this normalization may be omitted since the vectors are not biased by the collection occurrence frequencies of the terms. In fact, in our experiments we did not see a significant change in the experimental results using one approach or the other.

The generated document context vectors $\overrightarrow{d'}_i = (c_{i1}, c_{i2}, \dots \dots \dots, c_{in})^T$ correspond to the centroid of the term context vectors of all terms belonging to the document. Thus, the value of concept c_k in the document context vector for document i is the average of the influences of term tk on all terms occurring in di. Also, if matrix T is the identity matrix (e.g. all terms are represented only by themselves), then the context vectors for the index terms are pair-wise orthogonal and the resulting vectors have the same direction than the normal term frequency vectors. Only the scaling is different. This means that in this case no term dependencies are considered and our model behaves in the same way as the classical vector space model. Considering this fact it is interesting to analyze different ways of using the term context vectors. If we set all non-diagonal elements in T to 0, then the model behaves like VSM. On the other hand, if we set the diagonal to 0, then we get the other extreme. That is, the influence of a term in a document is only determined by all other terms that also occur in the document. Leaving all elements in term context vectors as defined above gives an intermediate model, but equation gives more importance to the non-diagonal elements, thus the "other" terms, than equation (1). In the experiments we tried both equations with and without setting the diagonal elements in T to 0.

2.1.3. Document Retrieval

Once the indexing process has been completed and all document context vectors are stored in the system, it can receive queries. For each query, it computes a list that presents documents in order of relevance. To calculate the relevance of a document to a query we use the standard cosine similarity measure:

$$s(d_i, q) = s(\vec{d'}_i, \vec{q}) = \frac{\sum_{j=1}^{n} p_q(j) q_j \cdot p_d(j) c_{ij}}{\sqrt{\sum_{j=1}^{n} \left(p_q(j) q_j\right)^2} \sqrt{\sum_{j=1}^{n} \left(p_d(j) c_{ij}\right)^2}}$$

where $\vec{d'}_i = (c_{i1}, c_{i2}, \dots \dots \dots, c_{in})^T$ is the document context vector for document d_i and $\vec{q} = (q_1, q_2, \dots, q_n)^T$ is a vector representing the query. Furthermore, $p_q(j)$ and $p_d(j)$ are some term weights applied to the components of the query and documents vectors, respectively. Documents are presented in order of decreasing cosine similarity.

Query vectors may be computed in different ways. In our experiment we used three different approaches: i) simple term frequency vectors for representing queries, ii) binary query vectors that indicate the existence of terms in the query (the frequency of the terms is omitted), and iii) query context vectors obtained in the same way as document context vectors. The first approach is possible since the concept space is spanned overall index terms. From an interpretative point of view, when applying frequency vectors the term frequencies of the terms in the query are considered as values for the corresponding concepts in the concept space. In the second approach each query term represents just the existence of the corresponding concept in the query but all the concepts are considered equally important. This may be appropriate in some queries where words that occur more often do not have a high semantic value. There, the importance of such words is “filtered out”. In our experiments we found out that our model performs better on some collections, when binary query vectors are used. This was also experienced by Wong et al. (1987). The third query indexing approach transforms queries in the same way as documents and, thus, increases the number of concepts that represent a query. In our experiments this method works better than the other approaches in some cases. However, considering memory, storage and calculation time requirements, the use of query context vectors is costlier than the other methods. Using vectors based on simple term frequencies drastically reduces retrieval time, because query indexing is much shorter and, more importantly, because the query vectors are sparse. Depending on the number of index terms, a query context vector will possibly have several thousand non-zero elements, whereas a term frequency or binary vector will only have a couple of non-zero values.

Equation includes a length normalization of the elements in the query and document vectors. (As pointed out earlier, we use the Euclidean vector norm to describe the length of a vector.) It is well known that differences in the length of document vectors may lead to worse retrieval results. Using only the scalar product as a similarity measure, for example, would mean that longer documents would have a higher probability of matching terms in a query than shorter ones. Therefore, they will be ranked higher. This also applies to our model, albeit for a slightly different reason. The document context vectors are already quite similar in length. Actually, if the normalization of term context vectors is used in equation, their length is always less than or equal to 1. This is because the term context vectors used in the indexing process are normalized to a length of 1. In fact, the closer the meaning of the terms occurring in a document, the closer the length of its vector will be to one. Even though this is an apparently interesting property we did not study it in more detail. Another variation in the length is caused by the multiplication with term weights. The normalization of query vectors has no impact on ranking and, therefore, may be omitted. This is because for a given query the same normalization applies to all documents. Thus, equation may be simplified to:

$$s(d_i, q) = s(\vec{d}'_i, \vec{q}) = \frac{\sum_{j=1}^{n} p_q(j) q_j \cdot p_d(j) c_{ij}}{\sqrt{\sum_{j=1}^{n} \left(p_d(j) c_{ij}\right)^2}}$$

The only advantage of using this instead of that is the reduction in calculation time. This can be quite important in on-line retrieval systems, where the response time is an issue of concern.

2.1.3.1. Term Weights

It is well known in the IR community, that the use of appropriate term weights can considerably improve the retrieval performance of a system. In the vector space model, each element of the original document vectors, that is, of the columns in the document/term matrix can be multiplied by such weights. Many different term-weighting schemata have been proposed, e.g., the signal weight (Dennis, 1967) or the discrimination value (Salton & Yang, 1973). However, *Inverse Document Frequency* (IDF) is probably the most commonly used and also one of the most effective weighting schemas. The IDF weight of a term t_i is calculated as follows:

$$\mathrm{idf}(t_i) = \log_2\left(\frac{m}{\mathrm{df}(t_i)}\right) + 1$$

where *m* is the number of documents in the collection and $\mathrm{df}(t_i)$ is the document frequency of term t_i, that is, the number of documents, in which the term occurs. The reason for using IDF weights is that terms that occur only in very few documents are better discriminators than terms that occur in the majority of the documents and, therefore, should be weighted higher.

Term-weighting techniques can also be expected to improve retrieval performance in the context vector model. Indeed, this is the case, as the presented results will show. Taking advantage of the fact that concepts and terms have a direct correspondence in our model, we can use the standard *idf* weight as defined in equation. Moreover, the difference of the context vector model to the classical VSM allows us to define other term weights.

2.1.3.2 Deviation measures of terms across document vectors.

Even though the classical IDF weights consider the number of different documents in which a term occurs, they do not consider the differences in the number of times a term occurs in those documents. As argued for IDF weights, it seems obvious that the variations in the occurrence frequency of terms in the documents will also be important. For example, a term that occurs in all documents with very different frequency values in each one will be more important than a term occurring in all documents just once. The IDF values in both cases, however, will be 1. This leads to the conclusion that some kind of deviation measure that assesses the differences of the occurrence frequency of terms across all documents should improve retrieval performance.

The motivation for using weights based on deviation values can also be seen from another point of view. As compared to the document vectors from the original term/document matrix, the document context vectors calculated in it are not very sparse. This is because they are calculated using not only those terms that actually occur in the documents, but also other terms co-occurring with them in at least one textual unit in the collection. Therefore, the number of non-zero elements in a document context vector will be much higher than the number of different index terms in the document. Furthermore, the values of the elements of document context vectors are real and not natural numbers. Because of these two properties there is no direct correspondence of *idf* weights in the context vector model.

Vector space model or term vector model is an algebraic model for representing text documents After considering the weighting terms in document collection, we can calculate similarity value between queries and documents. Ranking of documents depend upon score of similarity value calculated. The most relevant documents are then retrieved based on these ranks.

2.2 CryptDB:

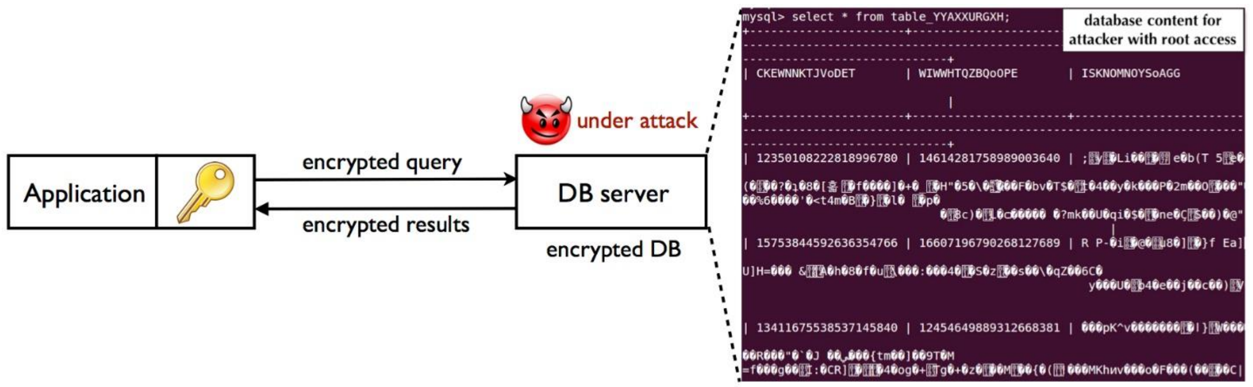
CryptDB is a DBMS that provides provable and practical privacy in the face of a compromised database server or curious database administrators. CryptDB works by executing SQL queries over encrypted data. At its core are three novel ideas: an SQL-aware encryption strategy that maps SQL operations to encryption schemes, adjustable query based encryption which allows CryptDB to adjust the encryption level of each data item based on user queries, and onion encryption to efficiently change data encryption levels. CryptDB only empowers the server to execute queries that the users requested, and achieves maximum privacy given the mix of queries issued by the users. The database server fully evaluates queries on encrypted data and sends the result back to the client for final decryption; client machines do not perform any query processing and client-side applications run unchanged.

CryptDB is a system that provides practical, provable confidentiality and privacy that guarantees without having to trust the DBMS server or the DBAs who maintain and tune the DBMS. CryptDB is the first private system to support all of standard SQL over encrypted data without requiring any client-side query processing, modifications to existing DBMS codebases, changes to legacy applications and offloads virtually all query processing to the server. CryptDB works by rewriting SQL queries, storing encrypted data in regular tables, and using an SQL user-defined function (UDF) to perform server-side cryptographic operations.

It works by executing SQL queries over encrypted data using a collection of efficient SQL-aware encryption schemes. CryptDB can also chain encryption keys to user passwords, so that a data item can be decrypted only by using the password of one of the users with access to that data. As a result, a database administrator never gets access to decrypted data, and even if all servers are compromised, an adversary cannot decrypt the data of any user who is not logged in.

2.2.1. System Architecture:

CryptDB comprises of three major components, namely the Application Server, Proxy Server and DBMS Sever. Application Server is the main server that runs CryptDB's database proxy and also the DBMS Server. The Proxy Server stores a secret Master Key, the database Schema and the onion layers of all the columns in the database. The DBMS server has access to the anonymized database schema, encrypted database data and also some cryptographic user-defined functions (UDF's). The DBMS server uses these cryptographic UDF's to carry out certain operations on the encrypted data (cipher text).



```
mysql> select * from table_YYAXXURGXH;
+----------------------+----------------------
-----------------------------------------------
----------------------------+
| CKEWNNKTJVoDET       | WIWWHTQZBQoOPE       | ISKNOMNOYSoAGG

                           |
+----------------------+----------------------+-----------------------
-----------------------------------------------------------------------
----------------------------+
| 12350108222818996780 | 14614281758989003640 | ;[illegible]Li��[illegible] e�b(T 5[illegible]
(�[illegible]�?�1�8�[[illegible]f����]�+� [illegible]H"�5�\�[illegible]��F�bv�T$�[illegible]�4��y�k���P�2m���O[illegible]���"
��%6����'�<t4m�B[illegible]}[illegible]l� [illegible]p�
                                   �[illegible]c)�[illegible]�c����� �?mk��U�qi�$�[illegible]ne�Ç[illegible]$��)�@"
                                                                |
| 15753844592636354766 | 16607196790268127689 | R P-�i[illegible]@�[illegible]u8�][illegible]}f Ea]
U]H=��� &[illegible]A�h�8�f�u[illegible]���:���4�[illegible]S�z[illegible]�s��\�qZ��6C�
                                                        y���U�[illegible]b4�e��j��c��)[illegible]


| 13411675538537145840 | 12454649889312668381 | ���pK^v�������[illegible]|}[illegible]���
��R���"�`�J ��,���{tm��]��9T�M
=f���g��[illegible]:�CR][illegible]4�og�+[illegible]g�+�z�[illegible]�M[illegible]�{�([illegible] ���MKhuv���o�F���(��[illegible]�C|
```

Fig 2.1 System Architecture

Below we describe the steps involved in processing a query inside CryptDB.

1. In the first step a query is issued by the application server. The proxy server receives this query and it anonymizes the table name and each of the column names. The proxy server also encrypts all the constants in the query using the stored secret master key. The encryption layers or the onion layers are also adjusted based on the type of operation required by the issued query. For example, if the query has to perform some equality checks then the deterministic encryption Scheme (DET)is applied to encrypt all the values in that particular column (on which equality check is to be performed).
2. The encrypted user query is then passed on to the DBMS server. The DBMS server executes these queries using standard SQL and also invokes UDF's to perform certain operations like token search and aggregation. The queries are executed on the encrypted database data.
3. The DBMS server performs computations on the encrypted data and forwards the encrypted results back to the proxy server.
4. The proxy server decrypts the encrypted query result obtained and returns it to the application server.

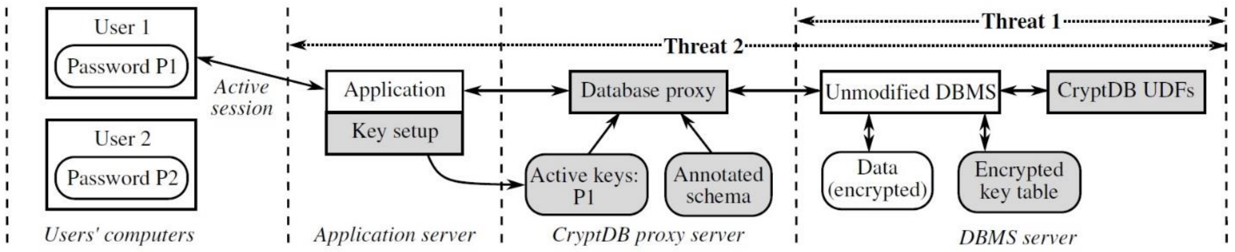


Fig 2.2 Threats

Threat 1: DBMS server compromise CryptDB provides confidentiality for the content of the data and for names of columns and tables, but does not hide the overall table structure, the number of rows, the types of columns, or the approximate size of data in bytes. The only information that CryptDB reveals to the DBMS server is relationships among data items corresponding to classes of computation that queries perform on the database, such as comparing items for equality, sorting, or performing word search. The granularity at which CryptDB allows the DBMS to perform a class of computations is an entire column (or a group of joined columns, for joins), which means that even if a query requires equality checks for a few rows, executing that query on the server would require revealing that class of computation for an entire column. CryptDB provides the following properties:

- ☐ Sensitive data is never available in plaintext at the DBMS server.
- ☐ If the application requests no relational predicate filtering on a column, nothing about the data content leaks (other than its size in bytes).
- ☐ If the application requests equality checks on a column, CryptDB's proxy reveals which items repeat in that column, but not the actual values.
- ☐ If the application requests order checks on a column, the proxy reveals the order of the elements in the column.

Threat 2: arbitrary confidentiality attacks on any servers The second threat is where the application server, proxy, and DBMS server infrastructures may be compromised arbitrarily. The approach in threat 1 is insufficient because an adversary can now get access to the keys used to encrypt the entire database. The solution is to encrypt different data items (e.g., data belonging to different users) with different keys. To determine the key that should be used for each data item, developers annotate the application's database schema to express finer-grained confidentiality policies. A curious DBA still cannot obtain private data by snooping on the DBMS server (threat 1), and in addition, an adversary who compromises the application server or the proxy can now decrypt only data of currently logged-in users (which are stored in the proxy). Data of currently inactive users would be encrypted with keys not available to the adversary, and would remain confidential. In this configuration, CryptDB provides strong guarantees in the face of arbitrary server-side compromises, including those that gain root access to the application or the proxy. CryptDB leaks at most the data of currently active users for the duration of the compromise, even if the proxy behaves in a Byzantine fashion. By "duration of a compromise", we mean the interval from the start of the compromise until any trace of the compromise has been erased from the system. For a read SQL injection attack, the duration of the compromise spans the attacker's SQL queries.

2.2.2. Onion Layers

When it comes to SQL aware encryption there are different aspects of computation that are based on different fundamental principles. For example, the operator GROUP BY relies on equality checks concerning the encrypted data, other functions like SUM rely on the ability to perform additions of the encrypted data. CryptDB deals with these different computational aspects by clustering functions by their underlying operations, as mentioned above. Around these different aspects or clusters CryptDB builds a construct that the developers have called onion: An onion features different layers of encryption from least revealing on the outside to most revealing on the inside (See fig 4.3). At the same time the outmost layer is the one with the least functionality while the innermost one offers the greatest functionality. The transformation from one layer into another ("peeling off a layer") happens automatically when the need arises (i.e. when a query with a certain operator/function is issued). In this case CryptDB automatically re-encrypts the entire column and remembers its state. While technically it is possibly to re-encrypt everything to a higher layer of security again it is not recommended by the developers in case of common queries as it would demand a considerable amount of computation power, besides that the information might have already been revealed.

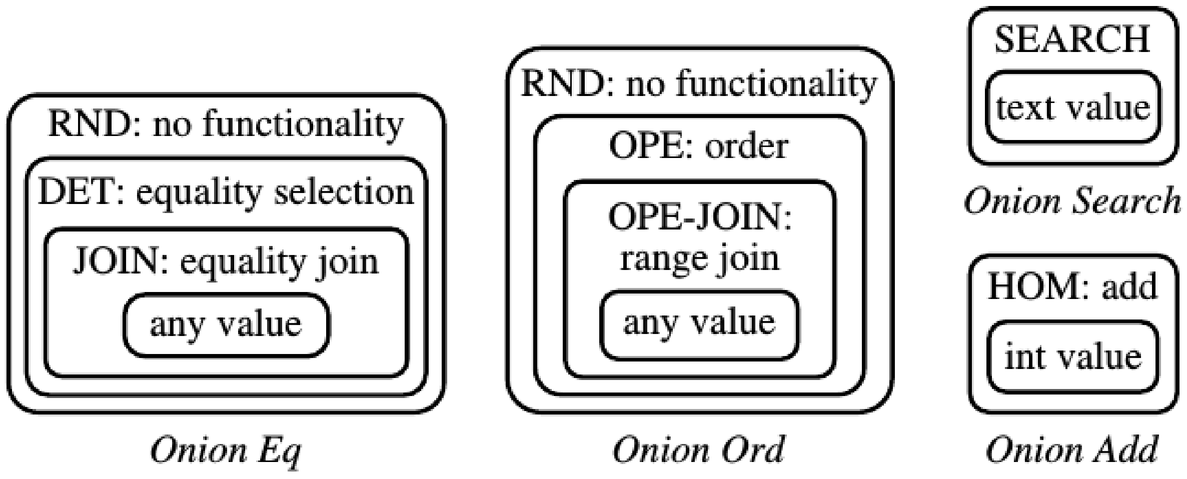


Fig 2.3 Onion Layers

2.2.3. Encryption Types

Each type uses a different algorithm that meets the specified requirements for a certain type and can be exchanged for another algorithm should the need arise, e.g. when a used cipher is broken. In such an event existing encrypted data would have to be decrypted with the old algorithm and re-encrypted using the new one. We have listed the different layers from most to least secure. Whereas least secure means that this particular layer does reveal the most information about its encrypted content, please notice that this is sometimes unavoidable in order to perform certain operations and is not automatically insecure.

2.2.3.1. Random (RND)

The RND onion layer provides the strongest security assurances: It is probabilistic, meaning that the same plaintext will be encrypted to a different cipher text. On the other hand, it does not allow any feasible computation in a reasonable amount of time. If someone wants to know something about the content of these fields the encrypted data has to be retrieved as a whole to be decrypted by CryptDB. This type seems to be reasonable choice for highly confidential data like medical diagnosis, private messages or credit card numbers that do not need to be compared to other entries for equality.

2.2.3.2. Homomorphic encryption (HOM)

The HOM onion layer provides an equally strong security assurance, as it is considered to be IND-CPA secure too [1]. It is specifically designed for columns of the data type integer and allows the database to perform operations of an additive nature. This includes of course the addition of several entries, but also operations like SUM or AVG. The reason that only addition is supported lies in the fact that fully homomorphic calculations, while mathematically proven by M. Cooney, is unfeasible slow on current hardware. An exception is the homomorphic addition $HOM(x) \cdot HOM(y) = HOM(x + y) \bmod n$, that can be performed in a reasonable amount of time. In CryptDB the developers choose to implement the homomorphic addition using the Paillier cryptosystem [13].

Currently the cipher text of a single integer is stored a VARBINARY (256), this means it uses 256 bytes of space which is 64 times the size of a normal integer that would only use 4bytes. Considering that integers are among the most used datatypes in a database. This is a huge overhead. Popa et al. indicate that there might be a more efficient way to store the integers with the use of a scheme developed by Ge and Zdonik. As of today this has not been implemented.

2.2.3.3. Word search (SEARCH)

The SEARCH onion layer is exclusive for columns of the data type text. In the version of CryptDB that we used in this thesis. We have been unable to successfully create such an onion. The following explanation is therefore solely of a theoretical nature and based on the paper provided by Popa et al. This layer uses a modified Version of a cryptographic scheme presented by Song et al. and allows for a keyword level text search with the LIKE operator. The implementation splits the string that is to be stored in the database by a specified delimiter (e.g. space or semicolon) and stores each distinct substring in a concatenated and encrypted form in the database. Each substring is padded to a certain size and its position inside the concatenated string is permutated thus obfuscating the position where it appears in the original string. When the user wants to perform a search using the LIKE operator CryptDB applies the padding to the search term and sends the encrypted version to the DBMS. The DBMS can now search for this specific string and is able to return the results. This scheme comes with several restrictions: Due to the used scheme it is only able to search for the existence of full words, it does not work with regular expressions or wildcards since they would not be encrypted in the same way.

2.2.3.4. Deterministic (DET)

The DET onion layer provides the second strongest security assurance: In contrary to RND this layer is deterministic, meaning that the same plaintext will be encrypted to the same cipher text. This means that the DBMS can identify fields with equal (encrypted) content. This allows us to use functions like GROUP BY, to group identical fields together or use DISTINCT to only select fields that are different. It does not however reveal whether a certain field is bigger or smaller than another field. For this type the developers used Blowfish and AES again, although this time they do not distinguish between integers and strings, but choose the cipher depending of the block size of the plaintext. Blowfish is used for any plaintext that is smaller than 64bit and AES for any plaintext that is bigger than 64bit. In both cases the plaintext is padded up to fit the block size. A special situation occurs when the plaintext is longer than the standard 128 bit AES block size: In this case the plaintext is split into several blocks which are processed in a variant of AES-CBC-mask-CBC (CMC)mode that uses a zero IV. Popa et al. justifies these special steps because AES in normal CBC mode would reveal prefix equality for the first n blocks in case the first n 128 bit blocks are identical [1].

2.2.3.5. Order-preserving encryption (OPE)

The OPE onion layer is significantly weaker than the DET layer as it reveals the order of the different entries. This means that the DBMS knows relations like bigger and smaller, but also equality (without having to look at the Eq onion). This means that if $x < y$, then $OPE(x) < OPE(y)$, also if $x = y$, then $OPE(x) = OPE(y)$. This allows us to use ordered operations like MIN, MAX or ORDER BY. To achieve this functionality, the developers of CryptDB implemented an algorithm that was published by Boldyreva et al. and was inspired by the ideas suggested by Agrawal et al. In regards to security it is noteworthy that this onion layer is the most revealing one: It cannot fulfill the security definition of IND-CPA, as is shown by Boldyreva et al. Even more important it reveal snot just the order but also the proximity of the transformed values to an attacker. This behavior might be acceptable for some values (e.g. text), but might be an issue for others (e.g. financial data).

2.2.3.6. Join (JOIN, OPE-JOIN)

The JOIN and OPE-JOIN layers are both “sublayers” of DET respective of OPE. That means both of them feature the computational abilities of their “parent layer” (i.e. to distinguish whether a plaintext a is equal to plaintext b, respective knowing the order of the entries of a column). In addition to that this type works over multiple columns and allows to determine whether a plaintext in column a is equal to a plaintext in column b for JOIN and whether a plaintext in column a is bigger or smaller than a plaintext in column b for OPE-JOIN. Both operators work with multiple column allowing for constructs like: SELECT * FROM test_table WHERE name1=name2 AND name2=name3. In this case all 3 name columns will use the same deterministic parameters, with the consequence that the same plaintext will be encrypted in the same cipher text across all three columns. Therefore, it is more revealing than DET or OPE alone.

2.2.4. CryptDB security related papers

Even though security is not an official part of this thesis, security is still an important topic when it comes to usability and whether it is worth the additional coasts. One question we had in the beginning was whether a curious database administrator could still draw conclusions from the encrypted data sets and whether he would able to take advantage of that, either by getting interesting insights or by actually being able to manipulate things in a way that would gain him further access to data. For these questions we would like to feature the following two papers: The first one is “On the Difficulty of Securing Web Applications using CryptDB” by Ihsan H. Akin and Berk Sunar and the second one is “Inference Attacks on Property Preserving Encrypted Databases” by Muhammad Naveed, Seny Kamara and Charles V. Wright.

The First paper shows that the lack of authentication checks for the stored data enables a malicious database administrator to copy and/or exchange row entries so that he could achieve administration privileges in a web application if he manages to identify the correct table. Another interesting aspect, as the paper points out that as long as the database administrator is able to interact with the web application and is able to “produce” queries whose changes he can log inside the DBMS he will most likely be able to figure out certain relations (e.g. by creating a user/logging in he will most likely be able to figure out the user table and even his users row). At this point he could then try to exchange/copy existing entries to gain further privileges, while the encryption is technically not broken the web application might be exploited. It is to notice that this might be less of an issue if the application using the database is not publicly accessible.

On the other hand the second paper describes a direct attack against the encryption by way of trying to determine the plaintext value of a ciphertext. To do that they have used two commonly known attacks (frequency analysis and sorting attacks) and also developed two new attacks (lp-optimization and cumulative attack). All these attacks focus on values encrypted with the DET or OPE layer - the two most revealing layers CryptDB has to offer. The proposed attacks have been shown to be very effective when used with dense data that is on a limited range (e.g. a scoring from 0-100, or other relatively fixed scales) or the frequency of the data is guessable (e.g. access control groups like administrators, moderators and users). Their findings have however spun up a little controversy between the authors of this paper and Ada Popa, the first author of the CryptDB paper who claims that they have wrongfully used the DET and OPE layer for non-high entropy values (i.e. values that fulfill the above mentioned criteria). The problem here is that quite a lot operations (like =,!=,count, min, max, group by, order by,...) require the functionality only offered by one of these two layers. So the question, how these low entropy values should - if at all - be encrypted, still remains open.

2.2.5. Discussion and Limitations

CryptDB's design supports most relational queries and aggregates on standard data types, such as integers and text/varchar types. Numeric data of decimal type with p digits after the decimal can be mapped to integer types by multiplying the input by 10p. CryptDB can encrypt floating-point values, but cannot perform aggregations on floating-point values per se (however, if the values are converted to fixed-point, CryptDB can use HOM for aggregation). Additional operations can be added to CryptDB by extending its existing onions, or adding new onions for specific data types. For example, one could add spatial and multi-dimensional range queries using the protocols proposed by Shi et al. CryptDB has certain limitations. For example, it does not support both computation and comparison in the same predicate, such as WHERE salary > age*2+10. CryptDB can facilitate processing such a query, but it would require a little bit of processing on the frontend. To use CryptDB, the query can be rewritten into a subquery that selects a whole column, SELECT age*2+10 FROM . . ., which CryptDB computes using HOM, then re-encrypting the results in the frontend, creating a new column (call it aux) at the server consisting of the newly-encrypted values, and finally running the original query with the predicate WHERE salary > aux. The current CryptDB prototype does not support stored procedures or other user-defined functions on the server. Supporting stored procedures written in SQL should be straightforward, by having the frontend rewrite the SQL statements inside of the stored procedure as it would for any other query. On the other hand, CryptDB's design cannot support the execution of arbitrary user defined functions (not written in SQL) over encrypted data. Finally, our current prototype handles server-side auto-increment columns by leaving the column values in plaintext on the server. We believe this is an acceptable trade-off, since the server is anyway involved in choosing the auto-incremented value. A more privacy preserving scheme may involve using HOM to generate the auto incremented value, but HOM would not allow joins on that column (whereas auto-increment is commonly used for primary key columns that require joins). More generally, CryptDB allows columns that are not privacy-sensitive to be left unencrypted on the server, to reduce overhead and allow more general computations (such as arbitrary UDFs).

2.2.6. Implementation

The CryptDB proxy consists of a C++ library and a Lua module. The C++ library consists of a query parser; a query encryptor/rewriter, which encrypts fields or includes UDFs in the query; and a result decryption module. To allow applications to transparently use CryptDB, we used MySQL proxy [47] and implemented a Lua module that passes queries and results to and from our C++ module. CryptDB implementation consists of _18,000 lines of C++ code and _150 lines of Lua code, with another _10,000 lines of test code. CryptDB is portable and we have implemented versions for both Postgres 9.0 and MySQL 5.1. The initial Postgres-based implementation is described in an earlier technical report [39]. Porting CryptDB to MySQL required changing only 86 lines of code, mostly in the code for connecting to the MySQL server and declaring UDFs. As mentioned earlier, CryptDB does not change the DBMS; all server-side functionality are implemented with

UDFs and server-side tables. CryptDB's design, and to a large extent our implementation, should work on top of any SQL DBMS that supports UDFs.

2.2.7. Guidelines

The guidelines an administrator must follow to obtain the security guarantees of CryptDB.

Step 1: Mark as "sensitive" any column whose content you want to protect. The next two steps are optional, but recommended for performance. For these steps, the administrator should supply the set of query types ran in the application. The constants don't matter here, only the operations. For example, web applications typically have a set of queries hardcoded in them, which are run with different constants, based on user data. Applications often have a relatively small set of query types; for instance, a TPC-C benchmark issues about 30 types of queries.

Step 2: Provide the query set to CryptDB. CryptDB checks if all queries are supported, and if there are unsupported queries, it outputs the reason: which operations cannot be supported, and why. If there are no unsupported queries, we are done.

Step 3: Address each unsupported query/operation with one or both of these options:

Option 1: (No change to applications) Use Monomi to split queries automatically between server and proxy.

Option 2: (Modify the application) Determine if the unsupported operations can be removed from the application or can be handled within the application. Unsupported queries are of two kinds: queries that cannot be supported with any of the encryption schemes in CryptDB, and queries that cannot be supported because some column was marked "sensitive". We have found the second category to be rare. The administrator is expected to handle both types of queries in the same way.

Option 3: Run the part of the query that cannot be supported on the client-side, in the proxy, as recommended of Monomi is a system designed to automate this process, without requiring any change to the application. Monomi can split a query into server- and client-side execution subject to some constraints. Monomi can operate with different constraints, and it receives the list of constraints as input. The constraints specify the operations that can happen on the field at the server. These operations correspond to the entire encryption set for a non-sensitive field, or to the strong encryption schemes for a sensitive field. Monomi finds an optimal splitting of these queries to maximize performance.

Option 4: The administrator can use the reason reported by CryptDB to understand which operations are not supported; the admin then decides if those operations are needed, if the application can run those queries differently, or if the computation can happen in the application. Functionality for "sensitive" fields Although fields marked "sensitive" are restricted only to strong encryption schemes, CryptDB can still perform a wide range of queries on these fields. shows the various operations that the server can compute directly on such fields in the database. The server uses a set of non-standard encryption schemes in addition to the standard scheme RND. Moreover, the CryptDB paper discusses that with a small amount of in-proxy computation, even more queries can run efficiently on such fields.

Hence, the queries supported on sensitive-marked fields are:

1. Fetch, modify, insert, or delete data due to RND. SQL examples include: SELECT, UPDATE, DELETE, COUNT, and INSERT.
2. Sum due to HOM (Paillier). SQL examples include: sum, +.
3. Product due to HOM (ElGamal). SQL examples include *.
4. Equality operations due to DET and SEARCH. SQL examples include: =, !=, <>, IN, NOT IN, etc. The proxy chooses between DET and SEARCH depending on whether the values in the column of interest are unique (that is, they do not repeat within the column).
    (a) If the column is declared unique by the administrator, the proxy allows the server to use DET. Indexes on this column work as expected.
    (b) If the values in the column are not unique, the server uses SEARCH. If there is no index built on this column, the performance of using SEARCH is comparable to using DET because the server scans the column of interest

in both cases, and SEARCH is based on fast hardware-based AES instructions. In typical applications, by far, most of the columns do not have an index on them. The columns that have an index are typically either not sensitive or are unique (e.g., primary keys). If there is a sensitive column that is not unique and has an index on it, the server can still perform equality operations using SEARCH on this column; this operation will be slower because the index is not used.

5. Certain order operations, such as ORDER BY with no limit. For example, the CryptDB paper discusses the example of a query of the form: SELECT [..] WHERE [...] ORDER BY sensitive field. CryptDB can support this query efficiently by executing SELECT [..] WHERE [...] at the server, and performing the order on the result set at the proxy. This strategy is efficient because the proxy anyways needs to receive and process the result set, and sorting the result set is a proportional amount of work. The expensive operation (which runs over a larger amount of data), the filtering due to WHERE, happens at the server.
6. Arithmetic functions or other functions that are computed on the result set and do not perform filtering. These can be executed at the proxy with low performance overhead, again, because the proxy needs to do work only in the result set. SQL examples include FORMAT, SUBSTRING, sin, etc. elaborates on these schemes. A recent blog posting claims that CryptDB restricts sensitive fields to RND only, and no queries can ever run on such data—this statement is false, as shown by the list above. In fact, the operations supported on fields marked "sensitive" are qualitatively the same as the queries for a field to be encrypted that is not marked sensitive: the difference is only in which cases an equality or an order can run efficiently. For equality, almost all sensitive fields will be supported with a comparable efficiency to non-sensitive fields. The reason is that in a typical application most fields do not have indexes built on them, and out of those that have indexes, a significant fraction is either not sensitive or are unique (e.g., primary keys). For example, for the OpenEMR medical application, more than 95% of fields do not have indexes built on them. Another example from OpenEMR is the field

"SSN", which is unique per patient and can be encrypted and queried by equality using DET.

7. Regarding order operations, a non-trivial number of fields in real-world applications process ORDER BY with no limit. For example, for a large database server (sql.mit.edu) with >1000 databases and >128,000 data fields, out of the original 13,131 data fields doing some order operation, more than >4600 data fields use ORDER BY with no limit. Out of the presumed sensitive fields in this trace, only a fraction of 0.001 had some order computation performed on them, suggesting that CryptDB will perform in-proxy processing for very few fields. For fields not marked sensitive, the server can additionally run range queries and ORDER BY LIMIT at the server. If an ORDER BY limit is needed for a sensitive field, Monomi optimizes the way in which this query gets split between proxy and server computation: in some cases, these queries can remain approximately as fast as before, in other cases, these can be slower. a significant class of computations can be performed on strongly encrypted fields, and
8. Keeping sensitive fields encrypted with strong encryption schemes is likely to result in few unsupported fields, for which we proposed measures.

Fields not marked "sensitive" If an administrator does not mark a field as sensitive, CryptDB provides no security guarantees. In this case, CryptDB employs a "best-effort" approach using the most secure encryption scheme that supports queries on these fields at the database server. Hence, such fields could end up being encrypted with weak encryption schemes such as OPE and DET for non-unique values. It is well-known that such schemes can leak data when coupled with other information an attacker might have. The CryptDB publications acknowledge this fact, and as a result require the use of the "sensitive" annotation to avoid such leakage.

2.2.8. Installation and Configuration of CryptDB:

1 sudo apt-get update

2 sudo apt-get install git ruby

```
3       git clone -b public git://g.csail.mit.edu/cryptdb
4       cd cryptdb
5       sudo ./scripts/install.rb .
```

Exporting directory:

```
export EDBDIR=/home/ubuntu/cryptdb/
```

Running Proxy Server:

```
/home/ubuntu/cryptdb/bins/proxy-bin/bin/mysql-proxy --plugins=proxy --event-
threads=4 --max-open-files=1024 --proxy-lua-script=$EDBDIR/mysqlproxy/wrapper.lua
--proxy-address=54.208.233.38:3307 --proxy-backend-addresses=localhost:3306
```

To connect to CryptDB:

```
mysql -u root -pletmein -h 54.208.233.38 -P 3307
```

2.2.9. Conclusion

CryptDB, a practical and novel system for ensuring data privacy on an untrusted SQL DBMS server. CryptDB uses three novel ideas to achieve its goal: an SQL-aware encryption strategy, adjustable query-based encryption, and onion encryption. As part of CryptDB's SQL-aware encryption strategy, optimizations for existing cryptographic techniques are proposed, as well as a new cryptographic mechanism for private joins. The prototype of CryptDB requires no changes to application or DBMS server code, and uses user-defined functions to perform cryptographic operations inside an existing DBMS engine, including both Postgres and MySQL.

3. PROBLEM DESCRIPTION

3.1 Problem Description

Theft of sensitive private data is a significant problem. Database management systems (DBMSs) are an especially appealing target for attackers, because they often contain large amounts of private information. When individual users or enterprises store their sensitive data in a DBMS today, they must trust that the server hardware and software are uncompromisable, that the data center itself is physically protected, and assume that the system and database administrators (DBAs) are trustworthy. Otherwise, an adversary who gains access to any of these avenues of attack can compromise the entire database.

These stringent security requirements are also at odds with cost Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. saving measures such as the consolidation of DBMSs belonging to different business units into a common enterprise-wide IT infrastructure, moving databases into a public cloud, or outsourcing DBA tasks. In fact, "lack of trust" is an oft-quoted primary concern about moving data in database systems to more cost-effective cloud infrastructures. Moreover, thanks to high-profile thefts of social security identifiers, credit card numbers, and other personal information from various online databases, these concerns are increasingly being reflected in the law as well.

In this project we build an application over CryptDB, a practical relational DBMS that provides provable privacy guarantees without having to trust the DBMS server or the DBAs who maintain and tune the DBMS. In CryptDB, unmodified DBMS servers store all data in an encrypted format, and execute SQL queries over encrypted data without having access to the decryption keys. CryptDB works by intercepting and rewriting all SQL queries in a frontend to make them execute on encrypted data, by encrypting and decrypting all data, as well as changing some query operators, while preserving the overall semantics of the query.

3.2 Minimum Hardware Requirements

Processor	:	1.2 GHz 2007 Opteron or 2007 Xeon processor
Memory	:	600 MB of RAM
Minimum Hard Disk Space	:	4GB

3.3 Software Requirements

Operating system	:	Ubuntu Server 12.04 LTS
Language Used	:	Java 1.8
IDE	:	NetBeans 8.2
Database	:	CryptDB

4. SYSTEM DESIGN

4.1 UML Diagrams

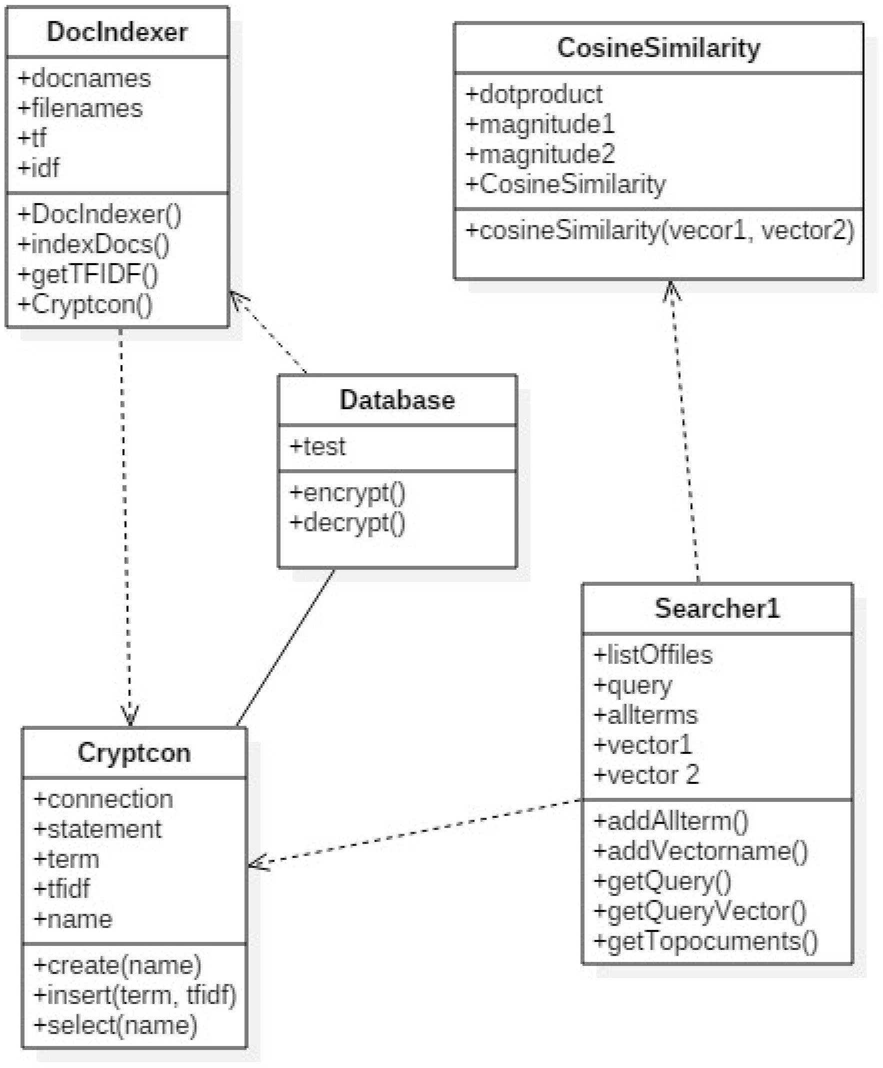


Fig 4.1 Class Diagram

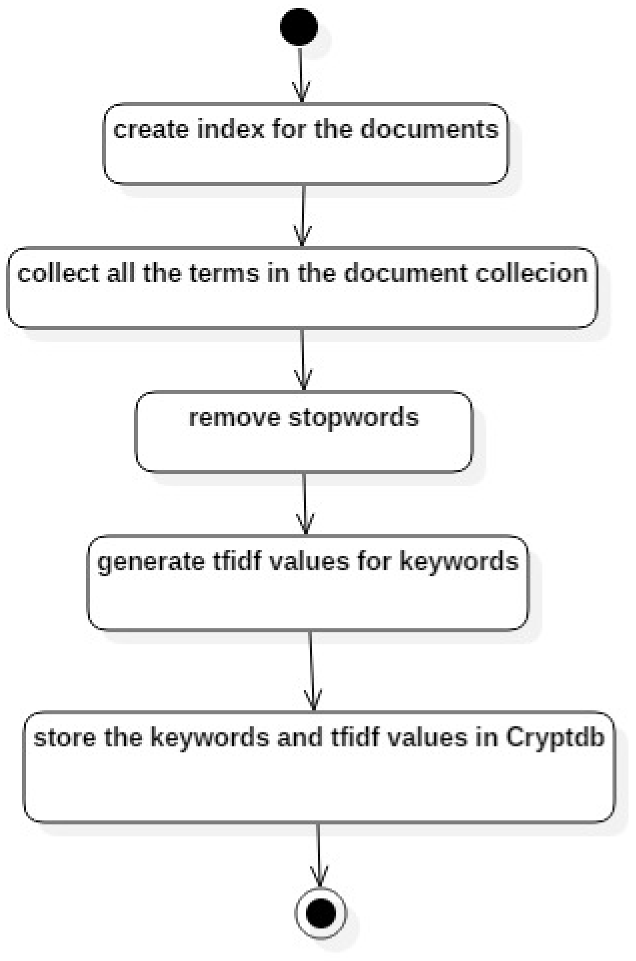

create index for the documents
collect all the terms in the document collecion
remove stopwords
generate tfidf values for keywords
store the keywords and tfidf values in Cryptdb

Fig 4.2. Activity Diagram 1

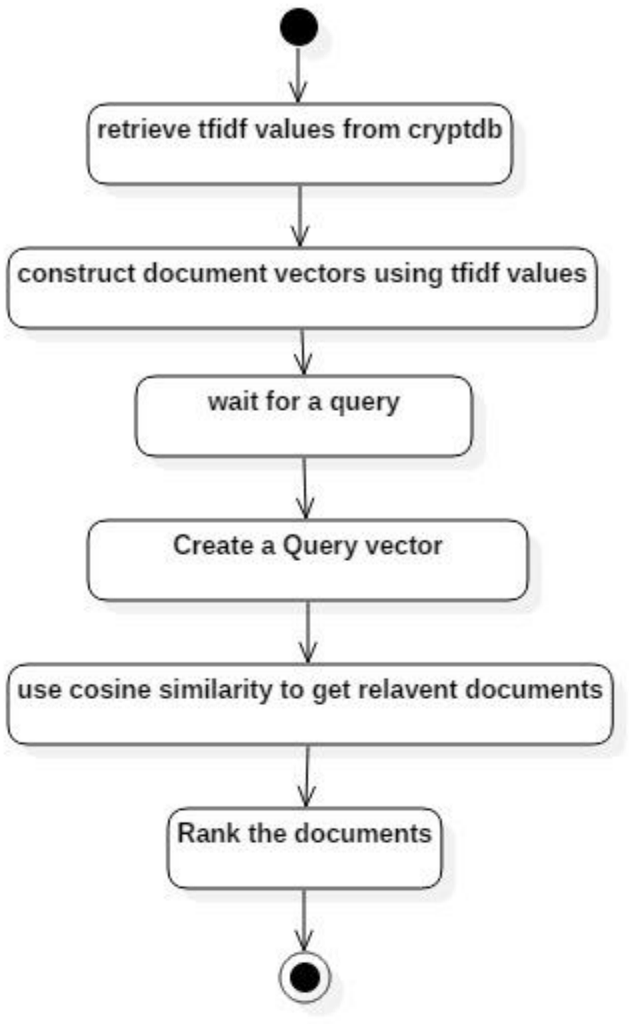

retrieve tfidf values from cryptdb
construct document vectors using tfidf values
wait for a query
Create a Query vector
use cosine similarity to get relavent documents
Rank the documents

Fig 4.3 Activity Diagram 2

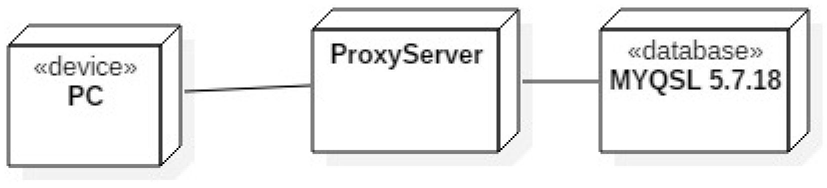


Fig 4.4 Deployment Diagram

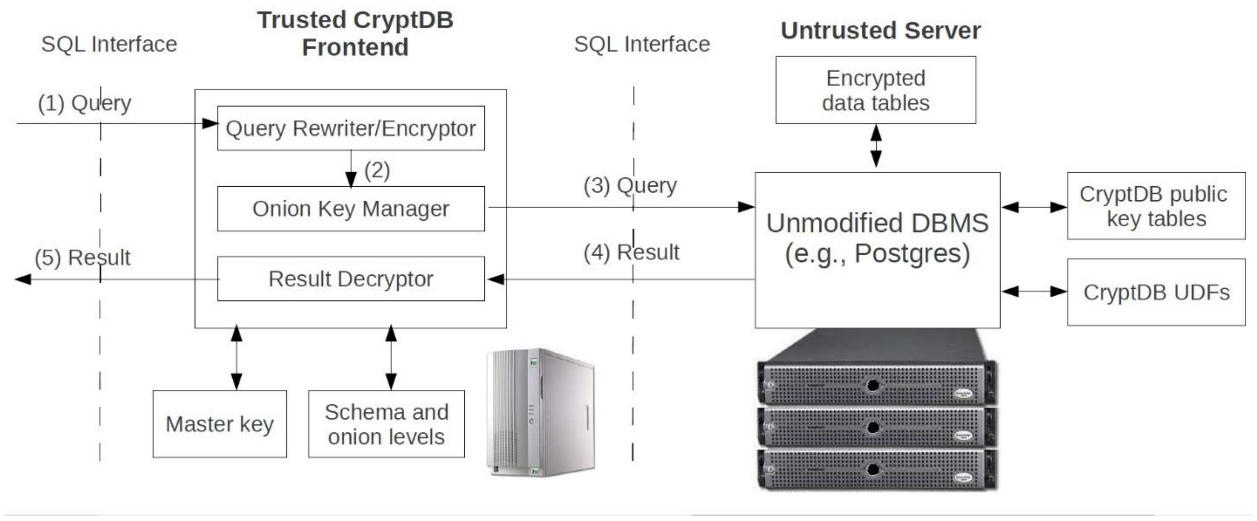


Fig 4.5 System Architecture

4.2 Introduction of Technologies used

About Java:

Initially the language was called as “oak” but it was renamed as “java” in 1995.The primary motivation of this language was the need for a platform-independent (i.e. architecture neutral) language that could be used to create software to be embedded in various consumer electronic devices.

- Java is a programming language.
- Java is cohesive and consistent.
- Except for those constraints imposed by the Internet environment. Java gives the programmer, full control.

Finally, Java is to Internet Programming where C was to System Programming.

Importance of Java to the Internet

Java has had a profound effect on the Internet. This is because; java expands the Universe of objects that can move about freely in Cyberspace. In a network, two categories of objects are transmitted between the server and the personal computer. They are passive information and Dynamic active programs. in the areas of Security and probability. But Java addresses these concerns and by doing so, has opened the door to an exciting new form of program called the Applet.

Applications and Applets

An application is a program that runs on our Computer under the operating system of that Computer. It is more or less like one creating using C or C++ .Java's ability to create Applets makes it important. An Applet is an application, designed to be transmitted over the Internet and executed by a Java-compatible web browser. An applet is actually a tiny Java program, dynamically downloaded across the network, just like an image. But the difference is, it is an intelligent program, not just a media file. It can be react to the user input and dynamically change.

Java Architecture

Java architecture provides a portable, robust, high performing environment for development. Java provides portability by compiling the byte codes for the Java Virtual Machine, which is then interpreted on each platform by the run-time environment. Java is a dynamic system, able to load code when needed from a machine in the same room or across the planet.

Compilation of code

When you compile the code, the Java compiler creates machine code (called byte code) for a hypothetical machine called Java Virtual Machine (JVM). The JVM is supposed t executed the byte code. The JVM is created for the overcoming the issue of probability. The code is written and compiled for one machine and interpreted on all machines .This machine is called Java Virtual Machine.

Compiling and interpreting java source code.

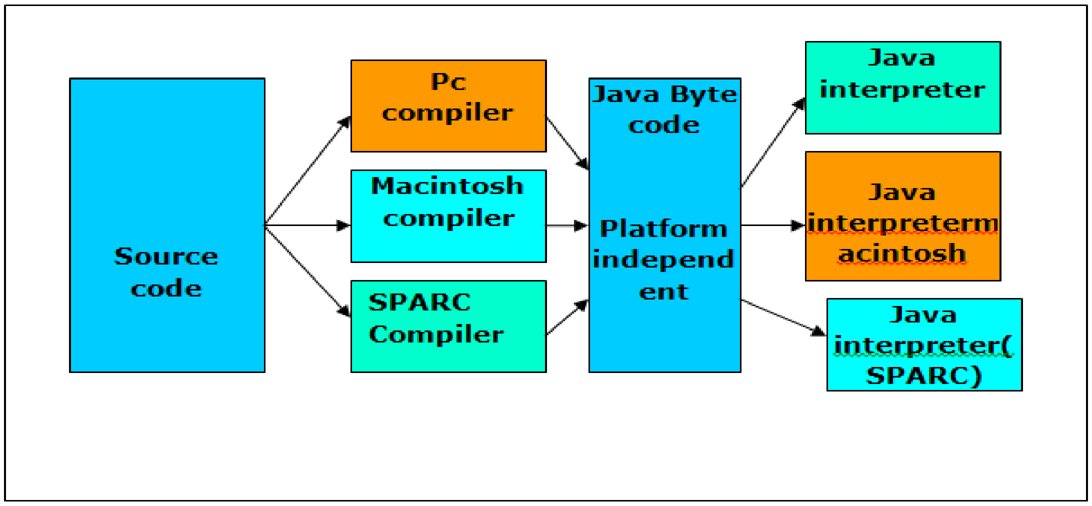


Fig 4.2.1 Compilation and Interpretation of Java

During run-time the Java interpreter tricks the byte code file into thinking that it is running on a Java Virtual Machine. In reality this could be an Intel Pentium windows 95 or sun SPARCstation running Solaris or Apple Macintosh running system and all could receive code from any computer through internet and run the Applets.

AWT:

Graphical User Interface:

The user interface is that part of a program that interacts with the user of the program. GUI is a type of user interface that allows users to interact with electronic devices with images rather than text commands. A class library is provided by the Java programming language which is known as Abstract Window Toolkit (AWT) for writing graphical programs. The Abstract Window Toolkit (AWT) contains several graphical widgets which can be added and positioned to the display area with a layout manager.

As the Java programming language, the AWT is not platform-independent. AWT uses system peers object for constructing graphical widgets. A common set of tools is provided by the AWT for graphical user interface design. The implementation of the user interface elements provided by the AWT is done using every platform's native GUI toolkit. One of the AWT's significance is that the look and feel of each platform can be preserved.

Components:

A graphical user interface is built of graphical elements called components. A *component* is an object having a graphical representation that can be displayed on the screen and that can interact with the user. Components allow the user to interact with the program and provide the input to the program. In the AWT, all user interface components are instances of class Component or one of its subtypes. Typical components include such items as buttons, scrollbars, and text fields.

Types of Components:

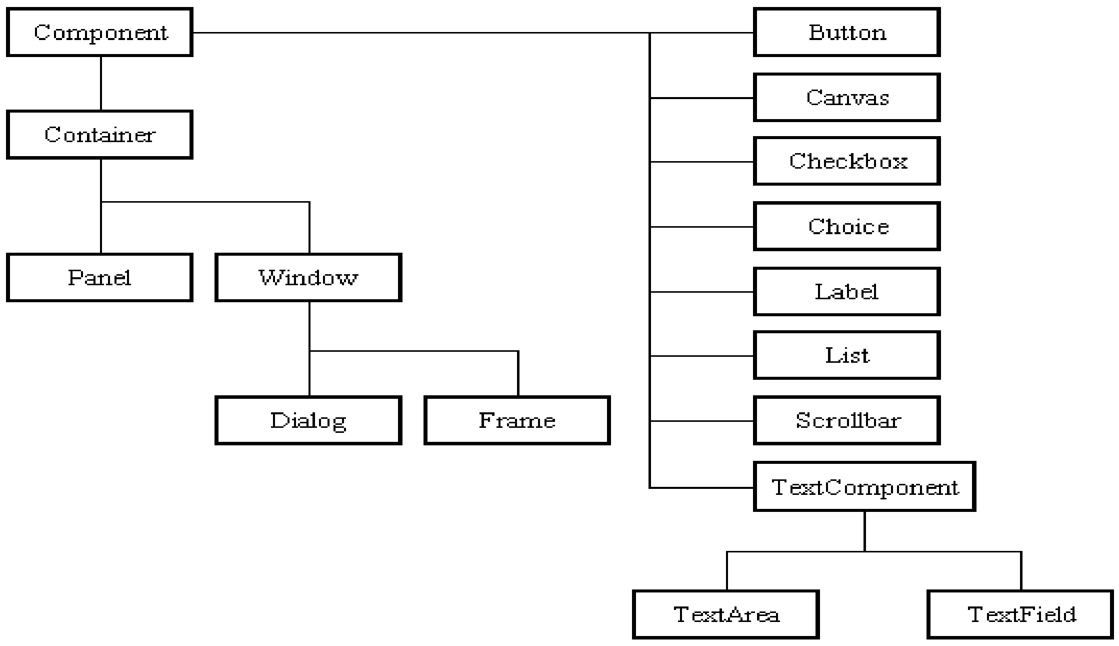


Fig 4.2.2 Types of Components

Before proceeding ahead, first we need to know what containers are. After learning containers we learn all components in detail.

<u>Containers:</u>

Components do not stand alone, but rather are found within containers. In order to make components visible, we need to add all components to the container. Containers contain and control the layout of components. In the AWT, all containers are instances of class Container or one of its subtypes. Components must fit completely within the container that contains them. For adding components to the container we will use add() method.

Types of containers:

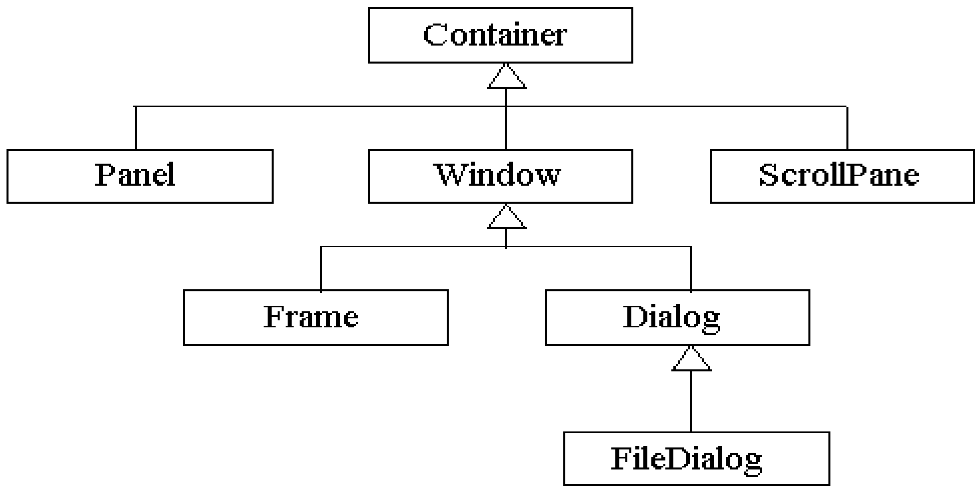


Fig 4.2.3 Types of Containers

<u>Basic GUI Logic:</u>

The GUI application or applet is created in three steps. These are:

- Add components to Container objects to make your GUI.
- Then you need to setup event handlers for the user interaction with GUI.
- Explicitly display the GUI for application.

A new thread is started by the interpreter for user interaction when an AWT GUI is displayed. When any event is received by this new thread such as click of a mouse, pressing of key etc then one of the event handlers is called by the new thread set up for GUI. One important point to note here is that the event handler code is executed within the thread.

Creating a Frame:

Method1:

In the first method we will be creating frame by extending Frame class which is defined in java.awt package. Following program demonstrate the creation of a frame.

```
import java.awt.*;

public class FrameDemo1 extends Frame

{

        FrameDemo1()

        {

                setTitle("Label Frame");

                setVisible(true);

                setSize(500,500);

        }

        public static void main(String[] args)

        {

                new FrameDemo1 ();

        }

}
```

In the above program we are using three methods:

setTitle: For setting the title of the frame we will use this method. It takes String as an argument which will be the title name.

SetVisible: For making our frame visible we will use this method. This method takes Boolean value as an argument. If we are passing true then window will be visible otherwise window will not be visible.

SetSize: For setting the size of the window we will use this method. The first argument is width of the frame and second argument is height of the frame.

Method 2:

In this method we will be creating the Frame class instance for creating frame window. Following program demonstrate Method2.

```
import java.awt.*;

public class FrameDemo2

{

        public static void main(String[] args)

        {

                Frame f = new Frame();

                f.setTitle("My first frame");

                f.setVisible(true);

                f.setSize(500,500);

        }

}
```

Types of Components:

1) Labels :

This is the simplest component of Java Abstract Window Toolkit. This component is generally used to show the text or string in your application and label never perform any type of action.

```
Label l1 = new Label("One");
Label l2 = new Label("Two");
Label l3 = new Label("Three",Label.CENTER);
```

In the above three lines we have created three labels with the name “one, two, three”. In the third label we are passing two arguments. Second argument is the justification of the label. Now after creating components we will be adding it to the container.

```
add(l1);
add(l2);
add(l3);
```

We can set or change the text in a label by using the **setText( )** method. You can obtain the current label by calling **getText( )**. These methods are shown here:

void setText(String *str*)

String getText( )

2) Buttons :

This is the component of Java Abstract Window Toolkit and is used to trigger actions and other events required for your application. The syntax of defining the button is as follows :

```
Button l1 = new Button("One");
Button l2 = new Button("Two");
Button l3 = new Button("Three");
```

We can change the Button's label or get the label's text by using the Button.setLabel(String) and Button.getLabel() method.

3) CheckBox:

A *check box is a control that is used to turn an option on or off. It consists of a small box* that can either contain a check mark or not. There is a label associated with each check box that describes what option the box represents. You change the state of a check box by clicking on it. The syntax of the definition of Checkbox is as follows:

```
Checkbox Win98 = new Checkbox("Windows 98/XP", null, true);

Checkbox winNT = new Checkbox("Windows NT/2000");

Checkbox solaris = new Checkbox("Solaris");

Checkbox mac = new Checkbox("MacOS");
```

The first form creates a check box whose label is specified in first argument and whose group is specified in second argument. If this check box is not part of a group, then *cbGroup* must be **null**. (Check box groups are described in the next section.) The value true determines the initial state of the check box is checked. The second form creates a check box with only one parameter.

To retrieve the current state of a check box, call **getState( )**. To set its state, call **setState( )**. You can obtain the current label associated with a check box by calling **getLabel( )**. To set the label, call **setLabel( )**. These methods are as follows:

```
boolean getState( )

void setState(boolean on)

String getLabel( )

void setLabel(String str)
```

Here, if *on* is **true**, the box is checked. If it is **false**, the box is cleared. The string passed in *str* becomes the new label associated with the invoking check box.

4) Radio Button:

This is the special case of the Checkbox component of Java AWT package. This is used as a group of checkboxes which group name is same. Only one Checkbox from a Checkbox Group can be selected at a time. Syntax for creating radio buttons is as follows:

```
CheckboxGroup cbg = new CheckboxGroup();

Checkbox Win98 = new Checkbox("Windows 98/XP", cbg , true);

Checkbox winNT = new Checkbox("Windows NT/2000",cbg, false);

Checkbox solaris = new Checkbox("Solaris",cbg, false);

Checkbox mac = new Checkbox("MacOS",cbg, false);
```

For Radio Button we will be using CheckBox class. The only difference in Checkboxes and radio button is in Check boxes we will specify null for checkboxgroup but whereas in radio button we will be specifiying the checkboxgroup object in the second parameter.

5) Choice:

The Choice class is used to create a pop-up list of items from which the user may choose. Thus, a Choice control is a form of menu. Syntax for creating choice is as follows:

```
Choice os = new Choice();

        /* adding items to choice */

        os.add("Windows 98/XP");

        os.add("Windows NT/2000");

        os.add("Solaris");

        os.add("MacOS");
```

We will be creating choice with the help of Choice class. Pop up list will be creating with the creation of object, but it will not have any items. For adding items we will be using add() method defined in Choice class.

To determine which item is currently selected, you may call either **getSelectedItem( )** or **getSelectedIndex( )**. These methods are shown here:

```
String getSelectedItem( )

int getSelectedIndex( )
```

The **getSelectedItem( )** method returns a string containing the name of the item. **getSelectedIndex( )** returns the index of the item. The first item is at index 0. By default, the first item added to the list is selected.

6) List:

List class is also same as choice but the only difference in list and choice is, in choice user can select only one item whereas in List user can select more than one item. Syntax for creating list is as follows:

```
List os = new List(4, true);
```

First argument in the List constructor specifies the number of items allowed in the list. Second argument specifies whether multiple selections are allowed or not.

```
/* Adding items to the list */

os.add("Windows 98/XP");

os.add("Windows NT/2000");

os.add("Solaris");

os.add("MacOS");
```

In list we can retrieve the items which are selected by the users. In multiple selection user will be selecting multiple values for retrieving all the values we have a method called getSelectedValues() whose return type is string array. For retrieving single value again we can use the method defined in Choice i.e. getSelectedItem().

7) TextField:

Text fields allow the user to enter strings and to edit the text using the arrow keys, cut and paste keys. TextField is a subclass of TextComponent. Syntax for creating list is as follows:

```
TextField tf1 = new TextField(25);
TextField  tf2 = new TextField();
```

In the first text field we are specifying the size of the text field and the second text field is created with the default value. **TextField** (and its superclass **TextComponent**) provides several methods that allow you to utilize a text field. To obtain the string currently contained in the text field, call **getText( )**. To set the text, call **setText( )**. These methods are as follows:

String getText( )

void setText(String *str*)

We can control whether the contents of a text field may be modified by the user by calling **setEditable( )**. You can determine editability by calling **isEditable( )**. These methods are shown here:

boolean isEditable( )

void setEditable(boolean *canEdit*)

isEditable( ) returns **true** if the text may be changed and **false** if not. In **setEditable( )**, if *canEdit* is **true**, the text may be changed. If it is **false**, the text cannot be altered.

There may be times when we will want the user to enter text that is not displayed, such as a password. We can disable the echoing of the characters as they are typed by calling **setEchoChar( )**.

8) TextArea:

TextArea is a multiple line editor. Syntax for creating list is as follows:

TextArea area = new TextArea(20,30);

Above code will create one text area with 20 rows and 30 columns. **TextArea** is a subclass of **TextComponent**. Therefore, it supports the **getText( )**, **setText( )**, **getSelectedText( )**, **select( )**, **isEditable( )**, and **setEditable( )** methods described in the preceding section.

TextArea adds the following methods:

void append(String *str*)

void insert(String *str*, int *index*)

void replaceRange(String *str*, int *startIndex*, int *endIndex*)

The **append( )** method appends the string specified by *str* to the end of the current text. **insert( )** inserts the string passed in *str* at the specified index. To replace text, call **replaceRange( )**. It replaces the characters from *startIndex* to *endIndex*–1, with the replacement text passed in *str*.

Layout Managers:

A layout manager automatically arranges controls within a window by using some type of algorithm. Each **Container** object has a layout manager associated with it. A layout manager is an instance of any class that implements the **LayoutManager** interface. The layout manager is set by the **setLayout( )** method. If no call to **setLayout( )** is made, then the default layout manager is used. Whenever a container is resized (or sized for the first time), the layout manager is used to position each of the components within it. The **setLayout( )** method has the following general form:

void setLayout(LayoutManager *layoutObj*)

Here, *layoutObj* is a reference to the desired layout manager. If you wish to disable the layout manager and position components manually, pass **null** for *layoutObj*. If we do this, you will need to determine the shape and position of each component manually, using the setBounds( ) method defined by Component.

Void setBounds(int x , int y , int width, int length)

In which first two arguments are the x and y axis. Third argument is width and fourth argument is height of the component.

Java has several predefined **LayoutManager** classes, several of which are described next. You can use the layout manager that best fits your application.

FlowLayout:

FlowLayout is the default layout manager. This is the layout manager that the preceding examples have used. **FlowLayout** implements a simple layout style, which is similar to how words flow in a text editor. Components are laid out from the upper-left corner, left to right and top to bottom. When no more components fit on a line, the next one appears on the next line. A small space is left between each component, above and below, as well as left and right. Here are the constructors for **FlowLayout**:

FlowLayout( )

FlowLayout(int *how*)

FlowLayout(int *how*, int *horz*, int *vert*)

The first form creates the default layout, which centers components and leaves five pixels of space between each component. The second form lets you specify how each line is aligned. Valid values for *how* are as follows:

FlowLayout.LEFT

FlowLayout.CENTER

FlowLayout.RIGHT

These values specify left, center, and right alignment, respectively. The third form allows you to specify the horizontal and vertical space left between components in *horz* and *vert*, respectively.

BorderLayout:

The **BorderLayout** class implements a common layout style for top-level windows. It has four narrow, fixed-width components at the edges and one large area in the center. The four sides are referred to as north, south, east, and west. The middle area is called the center. Here are the constructors defined by **BorderLayout**:

BorderLayout( )

BorderLayout(int *horz*, int *vert*)

The first form creates a default border layout. The second allows you to specify the horizontal and vertical space left between components in *horz* and *vert*, respectively. **BorderLayout** defines the following constants that specify the regions:

BorderLayout.CENTER	BorderLayout.SOUTH
BorderLayout.EAST	BorderLayout.WEST

BorderLayout.NORTH

When adding components, you will use these constants with the following form of **add( )**, which is defined by **Container**:

void add(Component *compObj*, Object *region*);

Here, *compObj* is the component to be added, and *region* specifies where the component will be added.

GridLayout:

GridLayout lays out components in a two-dimensional grid. When you instantiate a **GridLayout**, you define the number of rows and columns. The constructors supported by **GridLayout** are shown here:

GridLayout( )

GridLayout(int *numRows*, int *numColumns*)

GridLayout(int *numRows*, int *numColumns*, int *horz*, int *vert*)

The first form creates a single-column grid layout. The second form creates a grid layout with the specified number of rows and columns. The third form allows you to specify the horizontal and vertical space left between components in *horz* and *vert*, respectively. Either *numRows* or *numColumns* can be zero. Specifying *numRows* as zero allows for unlimited-length columns. Specifying *numColumns* as zero allows for unlimited-length rows.

4.3 MySQL

The MySQL™ software delivers a very fast, multi-threaded, multi-user, and robust SQL (Structured Query Language) database server. MySQL Server is intended for mission-critical, heavy-load production systems as well as for embedding into mass-deployed software. Oracle is a registered trademark of Oracle Corporation and/or its affiliates. MySQL is a trademark of Oracle Corporation and/or its affiliates, and shall not be used by Customer without Oracle's express written authorization. Other names may be trademarks of their respective owners.

The MySQL software is Dual Licensed. Users can choose to use the MySQL software as an Open Source product under the terms of the GNU General Public License (http://www.fsf.org/licenses/) or can purchase a standard commercial license from Oracle. See http://www.mysql.com/company/legal/licensing/ for more information on our licensing policies.

When commands are shown that are meant to be executed from within a particular program, the prompt shown preceding the command indicates which command to use. For example, shell> indicates a command that you execute from your login shell, root-shell> is similar but should be executed as root, and mysql> indicates a statement that you execute from the mysql client program:

```
shell> type a shell command here

root-shell> type a shell command as root here
```

```
mysql> type a mysql statement here
```

In some areas different systems may be distinguished from each other to show that commands should be executed in two different environments. For example, while working with replication the commands might be prefixed with `master` and `slave`:

```
master> type a mysql command on the replication master here

slave> type a mysql command on the replication slave here
```

The “shell” is your command interpreter. On Unix, this is typically a program such as **sh**, **csh**, or **bash**. On Windows, the equivalent program is **command.com** or **cmd.exe**, typically run in a console window.

When you enter a command or statement shown in an example, do not type the prompt shown in the example.

Database, table, and column names must often be substituted into statements. To indicate that such substitution is necessary, this manual uses ***db_name***, ***tbl_name***, and ***col_name***. For example, you might see a statement like this:

```
mysql> SELECT col_name FROM db_name.tbl_name;
```

This means that if you were to enter a similar statement, you would supply your own database, table, and column names, perhaps like this:

```
mysql> SELECT author_name FROM biblio_db.author_list;
```

SQL keywords are not case sensitive and may be written in any lettercase. This manual uses uppercase.

In syntax descriptions, square brackets (“[” and “]”) indicate optional words or clauses. For example, in the following statement, IF EXISTS is optional:

```
DROP TABLE [IF EXISTS] tbl_name
```

When a syntax element consists of a number of alternatives, the alternatives are separated by vertical bars (“|”). When one member from a set of choices *may* be chosen, the alternatives are listed within square brackets (“[” and “]”):

```
TRIM([[BOTH | LEADING | TRAILING] [remstr] FROM] str)
```

When one member from a set of choices *must* be chosen, the alternatives are listed within braces (“{” and “}”):

```
{DESCRIBE | DESC} tbl_name [col_name | wild]
```

An ellipsis (...) indicates the omission of a section of a statement, typically to provide a shorter version of more complex syntax. For example, `SELECT ... INTO OUTFILE` is shorthand for the form of `SELECT` statement that has an `INTO OUTFILE` clause following other parts of the statement.

An ellipsis can also indicate that the preceding syntax element of a statement may be repeated. In the following example, multiple ***`reset_option`*** values may be given, with each of those after the first preceded by commas:

```
RESET reset_option [,reset_option] ...
```

Commands for setting shell variables are shown using Bourne shell syntax. For example, the sequence to set the CC environment variable and run the configure command looks like this in Bourne shell syntax:

```
shell> CC=gcc ./configure
```

If you are using csh or tcsh, you must issue commands somewhat differently:

```
shell> setenv CC gcc

shell> ./configure
```

MySQL can be built and installed manually from source code, but it is more commonly installed from a binary package unless special customizations are required. On most Linux distributions, the package management system can download and install MySQL with minimal effort, though further configuration is often required to adjust security and optimization settings.

Though MySQL began as a low-end alternative to more powerful proprietary databases, it has gradually evolved to support higher-scale needs as well. It is still most commonly used in small to medium scale single-server deployments, either as a component in a LAMP-based web application or as a standalone database server. Much of MySQL's appeal originates in its relative simplicity and ease of use, which is enabled by an ecosystem of open source tools such as phpMyAdmin. In the medium range, MySQL can be scaled by deploying it on more powerful hardware, such as a multi-processor server with gigabytes of memory.

There are, however, limits to how far performance can scale on a single server ('scaling up'), so on larger scales, multi-server MySQL ('scaling out') deployments are required to provide improved performance and reliability. A typical high-end configuration can include a powerful master database which handles data write operations and is replicated to multiple slaves that handle all read operations. The master server continually pushes binlog events to connected slaves so in the event of failure a slave can be promoted to become the new master, minimizing downtime. Further improvements in performance can be achieved by caching the results from database queries in memory using memcached, or breaking down a database into smaller chunks called shards which can be spread across a number of distributed server clusters.

4.4 Apache Lucene

Apache Lucene^{TM} is a high-performance, full-featured text search engine library written entirely in Java. It is a technology suitable for nearly any application that requires full-text search, especially cross-platform.

Apache Lucene is an open source project available for free download.

4.4.1 Scalable, High-Performance Indexing

- Over 150GB per hour on modern hardware
- small RAM requirements -- only 1MB heap
- incremental indexing as fast as batch indexing
- index size roughly 20-30% the size of text indexed

4.4.2 Powerful, Accurate and Efficient Search Algorithms

- ranked searching -- best results returned first
- many powerful query types: phrase queries, wildcard queries, proximity queries, range queries and more
- fielded searching (e.g. title, author, contents)
- sorting by any field
- multiple-index searching with merged results
- allows simultaneous update and searching
- flexible faceting, highlighting, joins and result grouping

- fast, memory-efficient and typo-tolerant suggesters
- pluggable ranking models, including the Vector Space Model and Okapi BM25
- configurable storage engine (codecs)

4.4.3 Cross-Platform Solution

- Available as Open Source software under the Apache License which lets you use Lucene in both commercial and Open Source programs
- 100%-pure Java
- Implementations in other programming languages that are index-compatible

4.5. Implementation

4.5.1 DocIndexer.java

```
public class DocIndexer{
public DocIndexer(String path for index, String path to document collection)
{
        // input the documents
}
indexDocs()
{
          // create the indexes of the documents
}




getTfidf()
{
          //calculate the tfidf values of every term in the document collection
}
}
```

4.5.2 Cryptcon.java

```
public class Cryptcon
{
public Cryptcon()
{
        //establishing connection to the cloud server
}
public void create()
{
        //create a table for each document in the document Collection
}
public void insert()
{
        //for inserting terms abd their corresponding tfidf values into cryptdb
}
public void select()
{
        //for retreiving tfidf values from the corresponding table
}
}
```

4.5.3 CosineSimilarity.java

```
public class CosineSimilarity
{
public double cosineSimilarity(ArrayList<Integer> docVector1, ArrayList<Integer>
docVector2)
{
       //returns the dot product between two vector
}
}
```

4.5.4 Searcher.java

```
public class Searcher{
public void addallTerms()
{
        //collecting all the terms of the document collection from CryptDB
}
public void addvecName()
{
        //create a vector for every document
}
public void getQuery()
{
        //get the query from user
}
 public void getQueryVector()
{
        //Create a vector for query
}
public void getTopDocuments()
{
        //calculates the similarity between query and documents, ranks the documents and
        prints it
}
}
```

5. TESING AND RESULTS

Implementation and Testing:

Implementation is one of the most important tasks in project is the phase in which one has to be cautions because all the efforts undertaken during the project will be very interactive. Implementation is the most crucial stage in achieving successful system and giving the users confidence that the new system is workable and effective. Each program is tested individually at the time of development using the sample data and has verified that these programs link together in the way specified in the program specification. The computer system and its environment are tested to the satisfaction of the user.

Implementation

The implementation phase is less creative than system design. It is primarily concerned with user training, and file conversion. The system may be requiring extensive user training. The initial parameters of the system should be modifies as a result of a programming. A simple operating procedure is provided so that the user can understand the different functions clearly and quickly. The different reports can be obtained either on the inkjet or dot matrix printer, which is available at the disposal of the user. The proposed system is very easy to implement. In general implementation is used to mean the process of converting a new or revised system design into an operational one.

5.1 Testing

Testing is the process where the test data is prepared and is used for testing the modules individually and later the validation given for the fields. Then the system testing takes place which makes sure that all components of the system property functions as a unit. The test data should be chosen such that it passed through all possible condition. Actually testing is the state of implementation which aimed at ensuring that the system works accurately and efficiently before the actual operation commence. The following is the description of the testing strategies, which were carried out during the testing period.

5.1.1 System Testing

Testing has become an integral part of any system or project especially in the field of information technology. The importance of testing is a method of justifying, if one is ready to move further, be it to be check if one is capable to with stand the rigors of a particular situation cannot be underplayed and that is why testing before development is so critical. When the software is developed before it is given to user to user the software must be tested whether it is solving the purpose for which it is developed. This testing involves various types through which one can ensure the software is reliable. The program was tested logically and pattern of execution of the program for a set of data are repeated. Thus the code was exhaustively checked for all possible correct data and the outcomes were also checked.

5.1.2 Module Testing

To locate errors, each module is tested individually. This enables us to detect error and correct it without affecting any other modules. Whenever the program is not satisfying the required function, it must be corrected to get the required result. Thus all the modules are individually tested from bottom up starting with the smallest and lowest modules and proceeding to the next level. Each module in the system is tested separately. For example the job classification module is tested separately. This module is tested with different job and its approximate execution time and the result of the test is compared with the results that are prepared manually. The comparison shows that the results proposed system works efficiently than the existing system. Each module in the system is tested separately. In this system the resource classification and job scheduling modules are tested separately and their corresponding results are obtained which reduces the process waiting time.

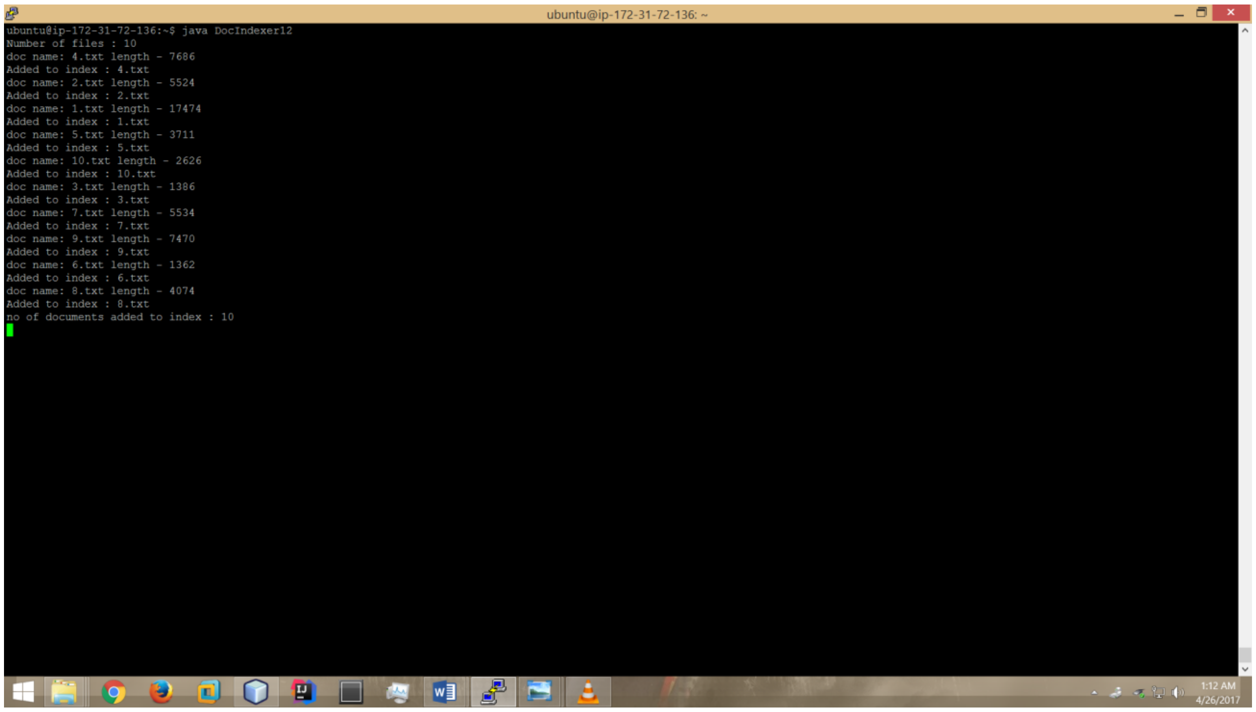
5.1.3 Integration Testing

After the module testing, the integration testing is applied. When linking the modules there may be chance for errors to occur, these errors are corrected by using this testing. In this system all modules are connected and tested. The testing results are very correct. Thus the mapping of jobs with resources is done correctly by the system.

5.1.4 Acceptance Testing

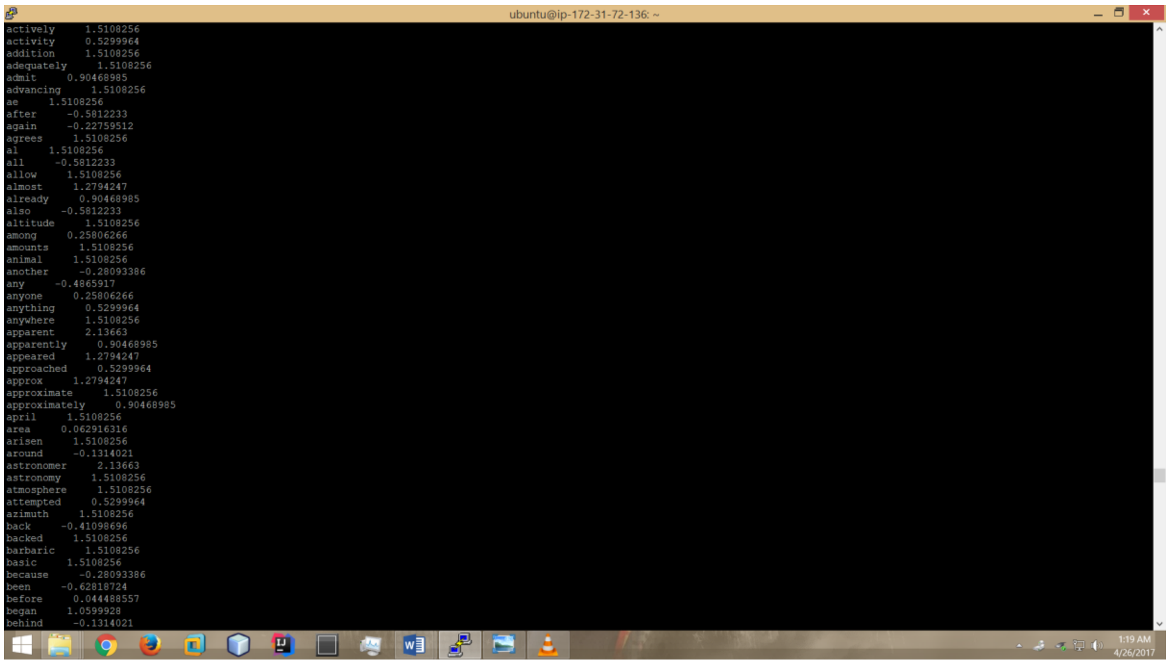
When that user fined no major problems with its accuracy, the system passers through a final acceptance test. This test confirms that the system needs the original goals, objectives and requirements established during analysis without actual execution which elimination wastage of time and money acceptance tests on the shoulders of users and management, it is finally acceptable and ready for the operation

5.2 SCREENSHOTS



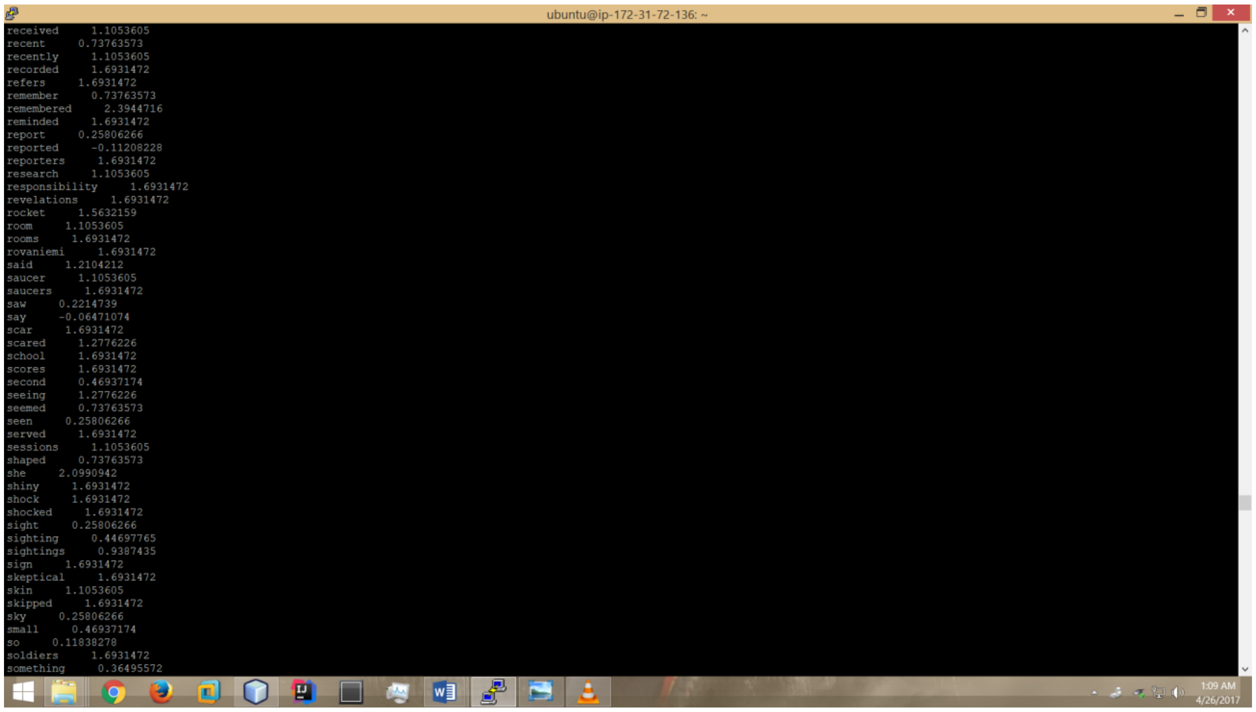
```
ubuntu@ip-172-31-72-136:~$ java DocIndexer12
Number of files : 10
doc name: 4.txt length - 7686
Added to index : 4.txt
doc name: 2.txt length - 5524
Added to index : 2.txt
doc name: 1.txt length - 17474
Added to index : 1.txt
doc name: 5.txt length - 3711
Added to index : 5.txt
doc name: 10.txt length - 2626
Added to index : 10.txt
doc name: 3.txt length - 1386
Added to index : 3.txt
doc name: 7.txt length - 5534
Added to index : 7.txt
doc name: 9.txt length - 7470
Added to index : 9.txt
doc name: 6.txt length - 1362
Added to index : 6.txt
doc name: 8.txt length - 4074
Added to index : 8.txt
no of documents added to index : 10
```

Fig 5.2.1 No of documents indexed



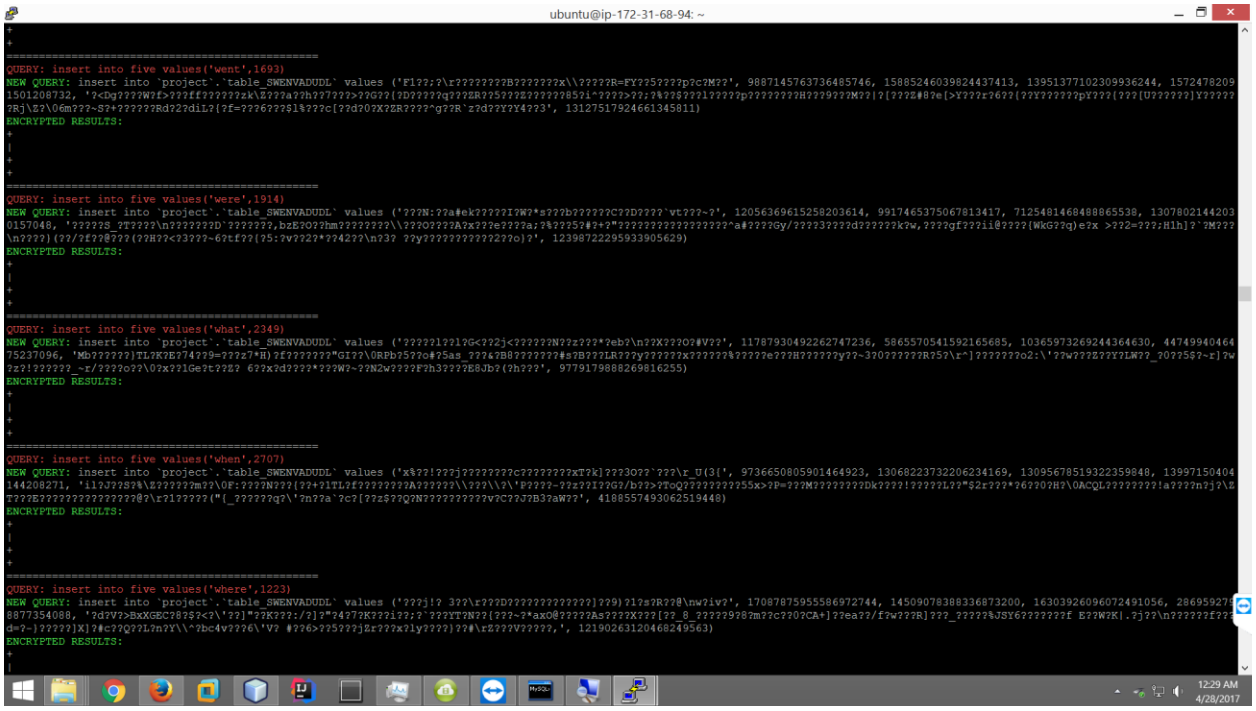
```
actively	1.5108256
activity	0.5299964
addition	1.5108256
adequately	1.5108256
admit	0.90468985
advancing	1.5108256
ae	1.5108256
after	-0.5812233
again	-0.22759512
agrees	1.5108256
al	1.5108256
all	-0.5812233
allow	1.5108256
almost	1.2794247
already	0.90468985
also	-0.5812233
altitude	1.5108256
among	0.25806266
amounts	1.5108256
animal	1.5108256
another	-0.28093386
any	-0.4865917
anyone	0.25806266
anything	0.5299964
anywhere	1.5108256
apparent	2.13663
apparently	0.90468985
appeared	1.2794247
approached	0.5299964
approx	1.2794247
approximate	1.5108256
approximately	0.90468985
april	1.5108256
area	0.062916316
arisen	1.5108256
around	-0.1314021
astronomer	2.13663
astronomy	1.5108256
atmosphere	1.5108256
attempted	0.5299964
azimuth	1.5108256
back	-0.41098696
backed	1.5108256
barbaric	1.5108256
basic	1.5108256
because	-0.28093386
been	-0.62818724
before	0.044488557
began	1.0599928
behind	-0.1314021
```

Fig 5.2.2 Indexing process 1



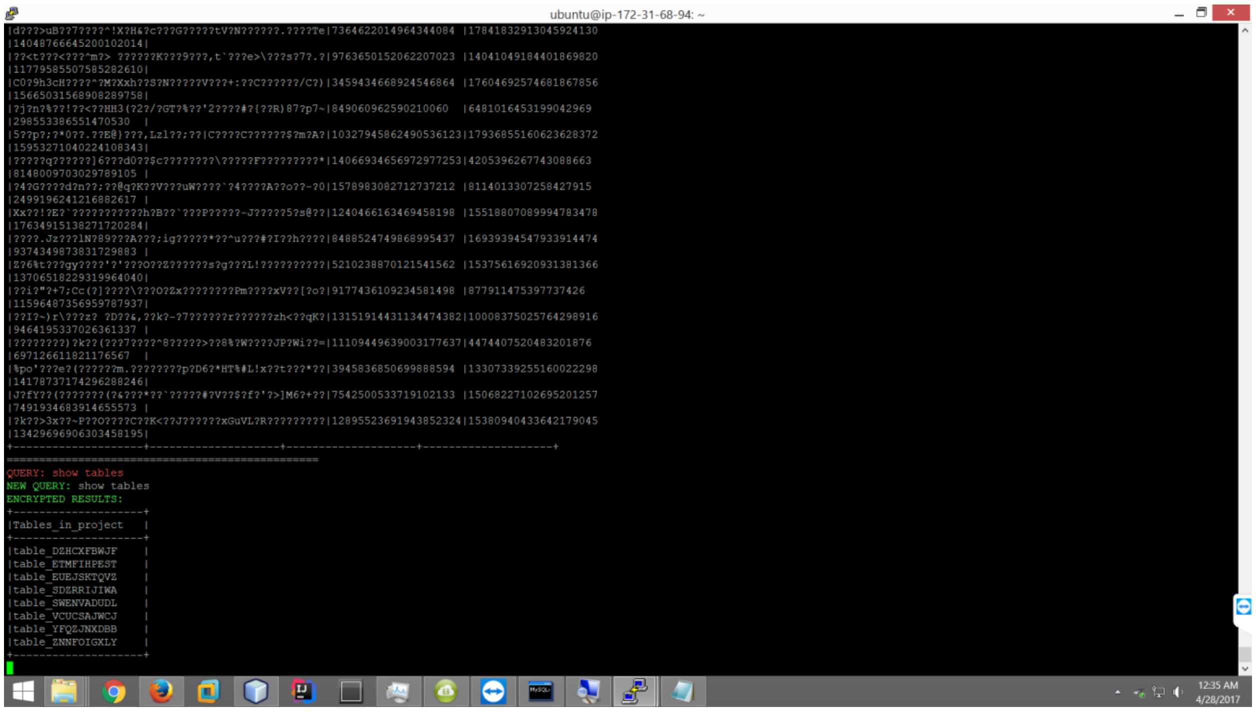
```
received	1.1053605
recent	0.73763573
recently	1.1053605
recorded	1.6931472
refers	1.6931472
remember	0.73763573
remembered	2.3944716
reminded	1.6931472
report	0.25806266
reported	-0.11208228
reporters	1.6931472
research	1.1053605
responsibility	1.6931472
revelations	1.6931472
rocket	1.5632159
room	1.1053605
rooms	1.6931472
rovaniemi	1.6931472
said	1.2104212
saucer	1.1053605
saucers	1.6931472
saw	0.2214739
say	-0.06471074
scar	1.6931472
scared	1.2776226
school	1.6931472
scores	1.6931472
second	0.46937174
seeing	1.2776226
seemed	0.73763573
seen	0.25806266
served	1.6931472
sessions	1.1053605
shaped	0.73763573
she	2.0990942
shiny	1.6931472
shock	1.6931472
shocked	1.6931472
sight	0.25806266
sighting	0.44697765
sightings	0.9387435
sign	1.6931472
skeptical	1.6931472
skin	1.1053605
skipped	1.6931472
sky	0.25806266
small	0.46937174
so	0.11838278
soldiers	1.6931472
something	0.36495572
```

Fig 5.2.3 Indexing Process 2



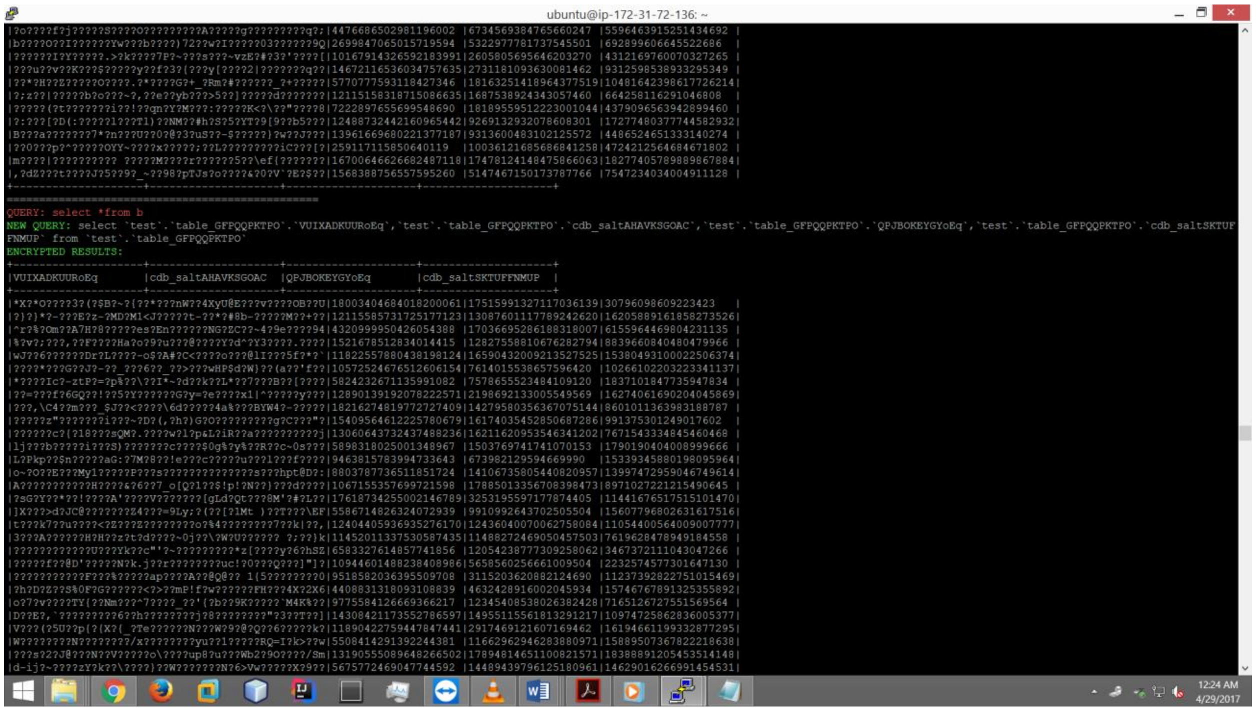
```
+
+
===============================================
QUERY: insert into five values('went',1693)
NEW QUERY: insert into `project`.`table_SWENVADUDL` values ('F1??;?\r????????B???????x\\?????R=FY??5????p?c?M??', 9887145763736485746, 15885246039824437413, 13951377102309936244, 1572478209
1501208732, '?<Dg????W?f>???ff??????zk\Z???a??h??7???>??G??{?D?????q???ZR??5???Z??????85?i^????>??;?%??$???l?????p????????H???9???M??|?[???Z#8?e[>Y???r?6??{??Y??????pY???{???[U??????]Y?????
?Rj\Z?\06m???~S?+??????Rd?2?diL?{?f=???6???$l%???c[??d?0?X?ZR????^g??R`z?d??Y?Y4??3', 13127517924661345811)
ENCRYPTED RESULTS:
+
|
+
+
===============================================
QUERY: insert into five values('were',1914)
NEW QUERY: insert into `project`.`table_SWENVADUDL` values ('???N:??a#ek?????I?W?*s???b??????C??D????`vt???~?', 12056369615258203614, 9917465375067813417, 7125481468488865538, 130780214420
0157048, '?????S_?T????\n???????D`???????,bzE?O??hm????????\\???O????A?x???e????a;?%???5?#?+?"??????????????????^a#????Gy/????3????d??????k?w,????gf???ii@????{WkG??q)e?x >???2=???;H1h]?`?M???
\n????}(??/?f??@???(??H??<?3???~6?tf??{?5:?v???*??42??\n?3? ??y?????????????o}?', 12398722295933905629)
ENCRYPTED RESULTS:
+
|
+
+
===============================================
QUERY: insert into five values('what',2349)
NEW QUERY: insert into `project`.`table_SWENVADUDL` values ('?????l??l?G<??2j<??????N??z???*?eb?\n??X???O?#V??', 11787930492262747236, 5865570541592165685, 10365973269244364630, 44749940464
75237096, 'Mb??????}TL?K?E?74??9=???z7*H)?f???????"GI??\0RPb?5??o#?5as_???&?B8???????#s?B???LR???y??????x??????%?????e???H??????y??~3?0??????R?5?\r^]???????o2:\'??w???Z??Y?LW??_?0??5$?~r]?w
?z?!??????_~r/????o??\0?x??1Ge?t??Z? 6??x?d????*???W?~??N2w????F?h3????E8Jb?(?h???', 9779179888269816255)
ENCRYPTED RESULTS:
+
|
+
+
===============================================
QUERY: insert into five values('when',2707)
NEW QUERY: insert into `project`.`table_SWENVADUDL` values ('x%??!???j????????c????????xT?k]???3O??`???\r_U(3{', 9736650805901464923, 13068223732206234169, 13095678519322359848, 13997150404
144208271, 'il?J??S?%\Z??????m??\0F:????N???{??+?1TL?f????????A??????\\???\\?\'P????-??z??I??G?/b??>?ToQ?????????55x>?P=???M????????Dk????!?????L??"$2r???*?6??0?H?\0ACQL????????!a????n?j?\Z
T???E????????????????@?\r?1?????("{_??????q?\'?n??a`?c?[??z$??Q?N??????????v?C??J?B3?aW??', 4188557493062519448)
ENCRYPTED RESULTS:
+
|
+
+
===============================================
QUERY: insert into five values('where',1223)
NEW QUERY: insert into `project`.`table_SWENVADUDL` values ('???j!? 3??\r???D?????????????]??9)?1?s?R??@\nw?iv?', 17087875955586972744, 14509078388336873200, 16303926096072491056, 28695927?
8877354088, '?d?V?>BxXGEC?8?$?<?\'??]"??K???:/?]?"?4?7?K???i??;?`???YT?N??{???~?*axO@?????As????X???[??_8_?????9?8?m??c??0?CA+]??ea??/f?w???R]???_?????%JSY6???????f E??W?K|.?j??\n??????f???
d=?-}?????]X]?#c??Q??L?n?Y\\^?bc4v???6\'V? #??6>??5???jZr???x?ly????}??#\rZ???V?????,', 12190263120468249563)
ENCRYPTED RESULTS:
+
|
```

Fig 5.2.4 Encrypting tfidf values in documents



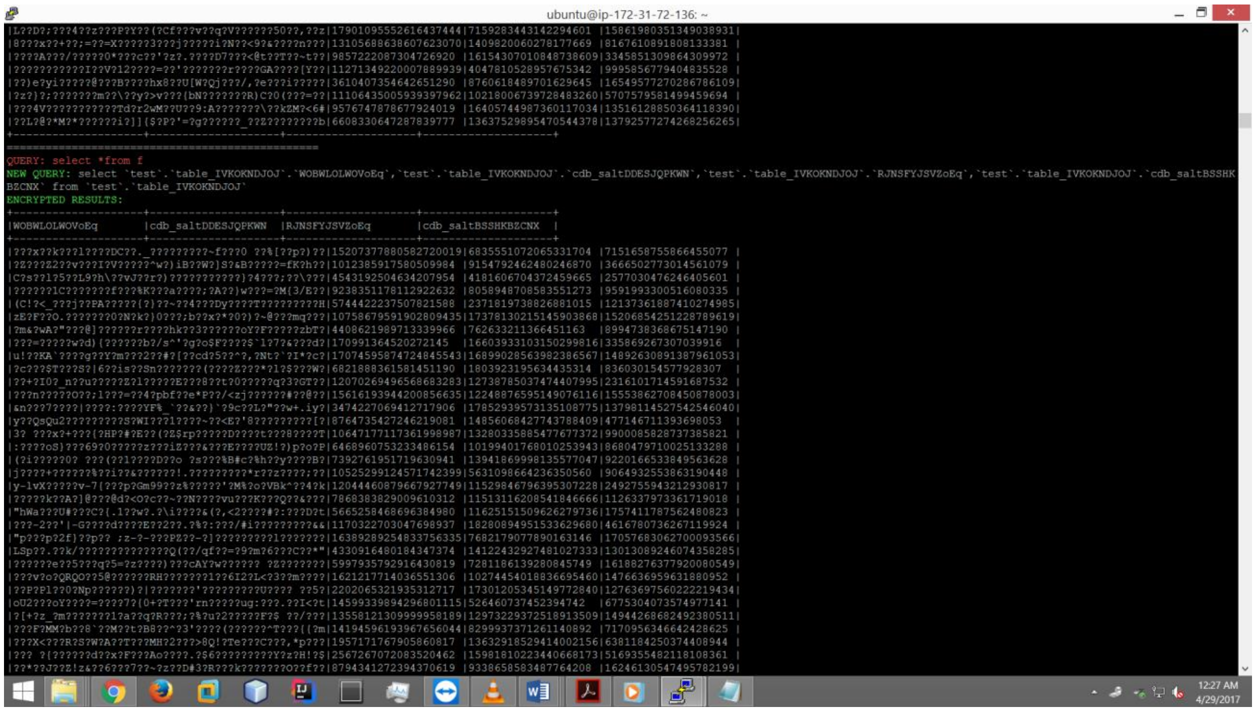
```
|d???>uB??7????^!X?H&?c???G?????tV?N??????.????Te|7364622014964344084 |17841832913045924130
|14048766645200102014|
|??<t???<???^m?> ??????K???9???,t`???e>\???s?7?.?|9763650152062207023 |14041049184401869820
|11779585507585282610|
|C0?9h3cH????^?M?Xxh??S?N?????V???+:??C??????/C?)|3459434668924546864 |17604692574681867856
|15665031568908289758|
|?j?n?%??!??<??HH3(?2?/?GT?%??'2????#?{??R)87?p7~|849060962590210060  |6481016453199042969
|2985533865514705 30  |
|5??p?;?*0??.??E@}???,Lzl??;??|C????C??????$?m?A?|10327945862490536123|17936855160623628372
|15953271040224108343|
|?????q??????]6???d0??$c????????\?????F?????????*|14066934656972977253|4205396267743088663
|8148009703029789105 |
|?4?G????d?n??;??@q?K??V???uW????`?4????A??o??-?0|1578983082712737212 |8114013307258427915
|2499196241216882617 |
|Xx??!?E?`??????????h?B??`???P?????-J?????5?s@??|1240466163469458198 |15518807089994783478
|17634915138271720284|
|????.Jz???lN?89???A???;ig?????*??^u???#?I??h????|8488524749868995437 |16939394547933914474
|9374349873831729883 |
|Z?6%t???gy????'?'???O??Z??????s?g???L!??????????|5210238870121541562 |15375616920931381366
|13706518229319964040|
|??i?"?+7;Cc(?]????\???O?Zx????????Pm????xV??[?o?|9177436109234581498 |877911475397737426
|11596487356959787937|
|??I?~)r\???z? ?D??&,??k?-?7??????r??????zh<??qK?|13151914431134474382|10008375025764298916
|9464195337026361337 |
|????????)?k??(???7????^8?????>??8%?W????JP?Wi??=|11109449639003177637|4474407520483201876
|697126611821176567  |
|%po'???e?(??????m.????????p?D6?*HT%#L!x??t???*??|3945836850699888594 |13307339255160022298
|14178737174296288246|
|J?fY??(???????(?&???*??`?????#?V??$?f?'?>]M6?+??|7542500533719102133 |15068227102695201257
|7491934683914655573 |
|?k??>3x??~P??O????C??K<??J??????xGuVL?R?????????|12895523691943852324|15380940433642179045
|13429696906303458195|
+--------------------+--------------------+--------------------+--------------------+
===============================================
QUERY: show tables
NEW QUERY: show tables
ENCRYPTED RESULTS:
+--------------------+
|Tables_in_project   |
+--------------------+
|table_DZHCXFBWJF    |
|table_ETMFIHPEST    |
|table_EUEJSKTQVZ    |
|table_SDZRRIJIWA    |
|table_SWENVADUDL    |
|table_VCUCSAJWCJ    |
|table_YFQZJNXDBB    |
|table_ZNNFOIGXLY    |
+--------------------+
```

Fig 5.2.5 List of tables in CryptDB



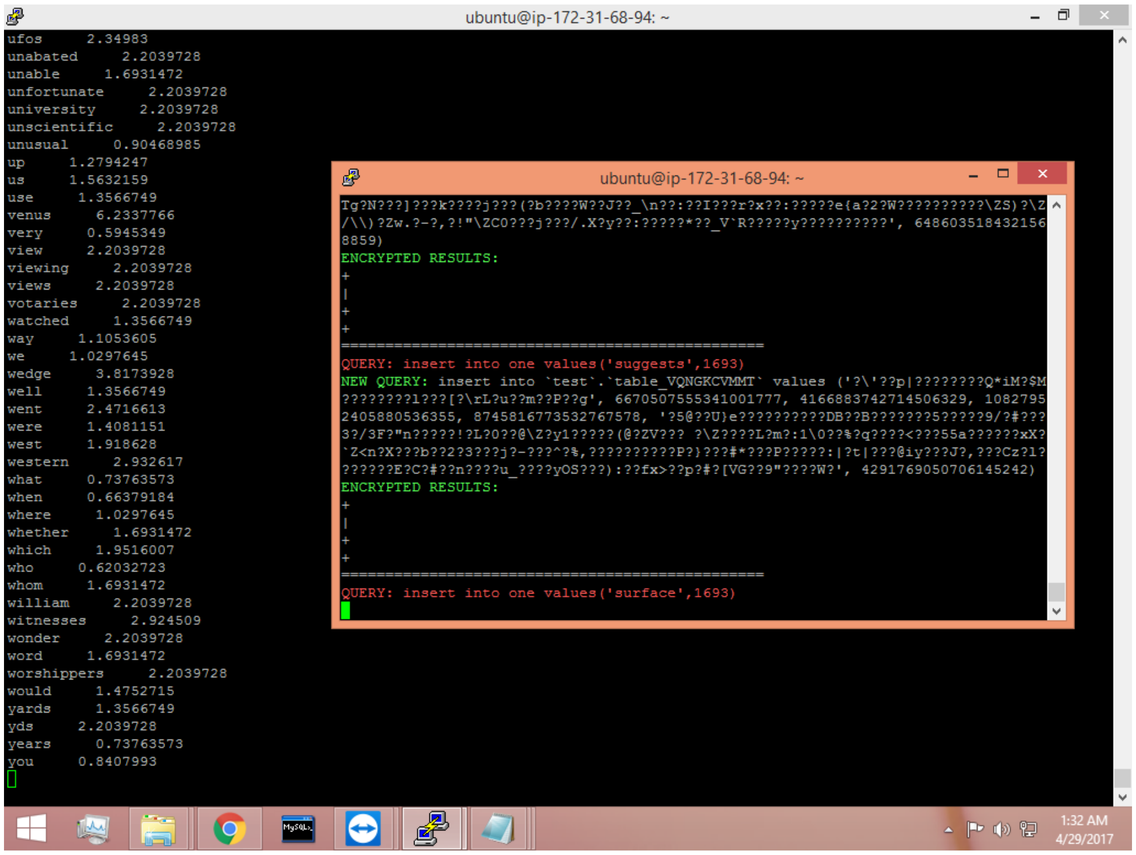
```
|?o????f?j?????S????O?????????A?????g?????????q?;|4476686502981196002 |6734569384765660247 |5596463915251434692 |
|b????O??I??????Yw???b????)72??w?I?????03???????9Q|2699847065015719594 |5322977781737545501 |6928996066645522686 |
|??????I?Y?????.>?k????7P?~???s???~vzE?#?3?'????[|10167914326592183991|2605805695646203270 |4312169760070327265 |
|???u??v??K???$?????y??f?3?(???y[?????2|???????q??|14672116536034757635|2731181093630081462 |9312598538933295349 |
|??*?H??Z?????O????.?*????G?+_?Rm?#??????_?+?????|5770777593118427346 |18163251418964377519|10481642398617726214|
|?;z??|?????b?o???~?,??e??yb???>5??]?????d???????|12115158318715086635|1687538924343057460 |6642581162910468O8 |
|?????(?t???????i??!??qn?Y?M???:?????K<?\??"????8|7222897655699548690 |18189559512223001044|4379096563942899460 |
|?:???[?D(:?????l???Tl)??NM??#h?S?5?YT?9[9??b5???|12488732442160965442|9269132932078608301 |17277480377744582932|
|B???a???????7*?n???U??0?@?3?uS??-$?????}?w??J???|13961669680221377187|9313600483102125572 |4486524651333140274 |
|??0???p?^?????OYY~????x?????;??L?????????iC???[?|2591171115850640119  |10036121685686841258|4724212564684671802 |
|m????|?????????? ?????M????r??????5??\ef(???????|16700646626682487118|17478124148475866063|18277405789889867884|
|,?dZ???t????J?5??9?_~??98?pTJs?o????&?0?V`?E?$??|1568388756557595260 |5147467150173787766 |7547234034004911128 |
+--------------------+--------------------+--------------------+--------------------+
================================================
QUERY: select *from b
NEW QUERY: select `test`.`table_GFPQQPKTPO`.`VUIXADKUURoEq`,`test`.`table_GFPQQPKTPO`.`cdb_saltAHAVKSGOAC`,`test`.`table_GFPQQPKTPO`.`QPJBOKEYGYoEq`,`test`.`table_GFPQQPKTPO`.`cdb_saltSKTUF
FNMUP` from `test`.`table_GFPQQPKTPO`
ENCRYPTED RESULTS:
+--------------------+--------------------+--------------------+--------------------+
|VUIXADKUURoEq       |cdb_saltAHAVKSGOAC  |QPJBOKEYGYoEq       |cdb_saltSKTUFFNMUP  |
+--------------------+--------------------+--------------------+--------------------+
|*X?*O????3?(?$B?~?{??*???nW??4XyU@E???v????OB??U|18003404684018200061|17515991327117036139|3079609860922342З   |
|?}?}*?-???E?z-?MD?M1<J?????t-??*?#8b-?????M??+??|12115585731725177123|13087601117789242620|16205889161858273526|
|^r?%?Om??A7H?8?????es?En??????NG?ZC??~4?9e????94|4320999950426054388 |17036695286188318007|6155964469804231135 |
|%?v?;???,??F????Ha?o?9?u???@????Y?d^?Y3????.????|15216785128340144l5 |12827558810676282794|8839660840480479966 |
|wJ??6??????Dr?L????-o$?A#?C<????o???@lI???5f?*?`|11822557880438198124|16590432009213527525|15380493100022506374|
|????*???G??J?-??_???6??_??>???wHP$d?W}??(a??'f??|10572524676512606154|7614015538657596420 |10266102203223341137|
|*????Ic?-ztP?=?p%??\??I*~?d??k??L*??7???B??[????|5824232671135991082 |7578655523484109120 |1837101847735947834 |
|??=???f?6GQ??!??5?Y??????G?y=?e????x1|^?????y???|12890139192078222571|2198692133005549569 |16274061690204045869|
|???,\C4??m???_$J??<????\6d?????4a%???BYW4?-?????|18216274819772727409|14279580356367075144|8601011363983188787 |
|?????z"???????i???~?D?(,?h?)G?O?????????g?C???"?|15409564612225780679|16174035452850687286|991375301249017602  |
|??????c?{?18???sQM?.????w?l?p&L?iR??a??????????j|13060643732437488236|16211620953546341202|7671543334845460468 |
|lj???b?????i???S)???????c????$0g%?y%??R??c~0s???|5898318025001348967 |1503769741741070153 |17901904040008999666|
|L?Pkp??$n?????aG:?7M?8??!e???c?????u???l???f????|9463815783994733643 |673982129594669990  |15339345880198095964|
|o~?O??E???My1?????P???s??????????????s???hpt@D?:|8803787736511851724 |14106735805440820957|13997472959046749614|
|A???????????H????&?6??7_o[Q?l??$!p!?N??}???d????|1067155357699721598 |17885013356708398473|8971027221215490645 |
|?sG?Y??*??!????A'????V???????[gLd?Qt???8M'?#?L??|17618734255002146789|3253195597177874405 |11441676517515101470|
|]X???>d?JC@???????Z4???=9Ly;?(??[?lMt )??T???\EF|5586714826324072939 |9910992643702505504 |15607796802631617516|
|t???k7??u????<?Z???Z????????o?%4??????????k|??,|12404405936935276170|12436040070062758084|11054400564009007777|
|3???A??????H?H??z?t?d????~0j??\?W?U?????? ?;??}k|11452011337530587435|11488272469050457503|7619628478949184558 |
|????????????U???Yk??c"'?~?????????*z[????y?6?hSZ|6583327614857741856 |12054238777309258062|3467372111043047266 |
|?????f??@D'?????N?k.j??r????????uc!?0???Q???]"]?|10944601488238408986|5658560256661009504 |2232574577301647130 |
|???????????F???%?????ap????A??@Q@?? 1{5?????????0|9518582036395509708 |3115203620882124690 |11237392822751015469|
|?h?D?Z??S%0F?G??????<?>??mP!f?w??????FH???4X?2X6|4408831318093108839 |4632428916002045934 |15746767891325355892|
|o??v????TY{??Nm???^7????_??'{?b??9K?????`M4K%??|9775584126669366217 |12345408538026382428|7165126727551569564 |
|D??E?,`?????????6??h????????j?8?????????"?3??T??]|14308421173552786597|14955115561813291217|10974725862836005377|
|V???(?5U??p{?(X?{_?Te??????N???W?9?@?Q??6?????k?|11890422759447847441|2917469121607169462 |16194661199332877295|
|W????????N????????/x????????yu??1?????RQ=I?k>??w|5508414291392244381 |11662962946283880971|15889507367822218638|
|???s?2?J@???N??V?????o\????up8?u???Wb2??O????/Sm|13190555089648266502|17894814651100821571|18388891205453514148|
|d-ij?~????zY?k??\????}??W???????N?6>Vw?????X?9??|5675772469047744592 |14489439796125180961|14629016266991454531|
```

Fig 5.2.6 Encrypted values in the database in table ‘b’



```
|L??D?;???4??z???P?Y??(?Cf???v??q?V??????5O??,??z|17901095552616437444|7159283443142294601 |15861980351349038931|
|8???x??+??;=??=X?????3???j?????i?N??<9?&????n???|13105688638607623070|1409820060278177669 |8167610891808133381 |
|????A???/?????0*???c??'?z?.????D7???<@t??T??~t??|9857222087304726920 |16154307010848738609|3345851309864309972 |
|???????????I??V?12????=??'???????r????GA????[Y??|11271349220007889939|4047810528957675342 |9995856779404835528 |
|??)e?yi?????@???B????hx8??U[W?Qj???/,?e???i?????|3610407354642651290 |8760618489701629645 |16549577270286786109|
|?z?}?;???????m??\??y?>v???(bN???????R)C?0(???=??|11106435005939397962|10218006739728483260|5707579581499459694 |
|???4V???????????Td?r2wM??U??9:A???????\??kZM?<6#|9576747878677924019 |16405744987360117034|13516128850364118390|
|??L?@?*M?*??????i?]]{$?P?'=?g??????_??Z????????b|6608330647287839777 |13637529895470544378|13792577274268256265|
+--------------------+--------------------+--------------------+--------------------+
================================================
QUERY: select *from f
NEW QUERY: select `test`.`table_IVKOKNDJOJ`.`WOBWLOLWOVoEq`,`test`.`table_IVKOKNDJOJ`.`cdb_saltDDESJQPKWN`,`test`.`table_IVKOKNDJOJ`.`RJNSFYJSVZoEq`,`test`.`table_IVKOKNDJOJ`.`cdb_saltBSSHK
BZCNX` from `test`.`table_IVKOKNDJOJ`
ENCRYPTED RESULTS:
+--------------------+--------------------+--------------------+--------------------+
|WOBWLOLWOVoEq       |cdb_saltDDESJQPKWN  |RJNSFYJSVZoEq       |cdb_saltBSSHKBZCNX  |
+--------------------+--------------------+--------------------+--------------------+
|???x??k???l????DC??._?????????~f???0 ??%[??p?)??|15207377880582720019|6835551072065331704 |7151658755866455077 |
|?Z???Z??v???I?V?????^w?)iB??W?]S?&B?????=fK?h??|1012385917580509984 |9154792462480246870 |3666502773014561079 |
|C?s??l?5??L9?h\??vJ??r?)???????????}?4???;??\???|4543192504634207954 |4181606704372459665 |2577030476246405601 |
|??????1C???????f???%K???a????;?A??)w???=?M(3/E??|9238351178112922632 |8058948708583551273 |9591993300516080335 |
|(C!?<_???j??PA?????{?}??~??4???Dy????T?????????H|5744422237507821588 |2371819738826881015 |12137361887410274985|
|zE?F??O.????????0?N?k?}0???;b??x?*?0?)?~@???mq???|10758679591902809435|17378130215145903868|15206854251228789619|
|?m&?wA?"???@]??????r????hk??3???????oY?F?????zbT?|4408621989713339966 |762633211366451163  |8994738368675147190 |
|???=?????w?d){??????b?/s^'?g?o$F????$`l?7?&???d?|170991364520272145  |16603933103150299816|3358692673O7039916  |
|u!??KA`????g??Y?m???2???#?[??cd?5??^?,?Nt?`?I*?c?|17074595874724845543|16899028563982386567|14892630891387961053|
|?c???$T???S?|6??is??Sn???????(????Z???*?l?$???W?|6821888361581451190 |1803923195634435314 |836030154577928307  |
|??+?IO?_n??u?????Z?l?????E???8??t?0?????q?3?GT??|12070269496568683283|12738785037474407995|2316101714591687532 |
|???n?????O??;l???=??4?pbf??e*P??/<zj??????#??@??|15616193944200856635|12248876595149076116|15553862708450878003|
|&n???7????|????:????YF%_`??&??}`?9c??L?"??w+.iy?|3474227069412717906 |17852939573135108775|13798114527542546040|
|y??QsQu2??????????S?WI???1????~??<E?'8?????????[?|8764735427246219081 |14856068427743788409|477146711393698053  |
|3? ???x?+???{?HP?#?E??(?Z$rp?????D????t???8????T|10647177117361998987|13280335885477677372|9900085828737385821 |
|:????oS}???69?0?????z???iZ???&???E????UZ!?)p?o?P|6468960753233486154 |10199401768010253943|8680479710025133288 |
|(?i?????0? ???(??l????D??o ?s???%B#c?%h??y????B?|7392761951719630941 |13941869998135577047|9220166533849563628 |
|j????+??????%??i??&??????!.?????????*r??z????;??|10525299124571742399|5631098664236350560 |9064932553863190448 |
|y-lvX?????v-7{???p?Gm99??z%?????'?M%?o?VBk^??4?k|12044460879667927749|11529846796395307228|2492755943212930817 |
|?????k??A?]@???@d?<O?c??~??N????vu???K???Q??&???|7868383829009610312 |11513116208541846666|11263379733617l9018 |
|"hWa???U#???C?{.1??w?.?\i????&(?,<2?????#?:???D?t|5665258468696384980 |11625151509626279736|1757411787562480823 |
|???-2??'|-G????d????E??2??.?%?:???/#i??????????&&|11703227030476989З7 |18280894951533629680|4616780736267119924 |
|"p???p??f}??p?? ;z-?-???PZ??-?]?????????1???????|16389289254833756335|7682179077890163146 |17057683062700093566|
|LSp??.??k/??????????????Q(??/qf??=?9?m?6???C??*"|4330916480184347374 |14122432927481027333|13013089246074358285|
|??????e??5???q?5=?z????)???cAY?w?????? ?Z???????|5997935792916430819 |7281186139280845749 |16188276377920080549|
|???v?o?QRQO??5@??????RH????????1??6I2?L<?3??m????|1621217714036551306 |10274454018836695460|1476636959631880952 |
|??P?Pl??0?Np??????)?|???????'?????????U???? ??5?|2202065321935312717 |17301205345149772840|12763697560222219434|
|oU2???oY????=?????7{0+?T???'rn?????ug:???.??I<?t|14599339894296801115|526460737452394742  |6775304073574977141 |
|?[+?z_?m????????1?a??q?R???;?%?u?2?????F?$ ??/???|13558121309999958189|12973229372518913509|14944268682492380511|
|???F?MM?b??8`??M??t?B8??^?3'????(??????^T???{(?m|14194596193967656044|8299937371261140892 |7170956346642428625 |
|???X<???R?S?W?A??T???MH?2???>8Q!?Te???C???,*p???|1957171767905860817 |13632918529414002156|6381184250374408944 |
|??? ?{??????d??x?F???Ao????.?$6?????????Y?z?H!?$|2567267072083520462 |15981810223440668173|5169355482118108361 |
|??*??J??Z!z&??6???7??~?z??D#3?R???k???????O??f??|8794341272394370619 |9338658583487764208 |16246130547495782199|
```

Fig 5.2.7 Encrypted values in the database in table ‘f’



```
ufos	2.34983
unabated	2.2039728
unable	1.6931472
unfortunate	2.2039728
university	2.2039728
unscientific	2.2039728
unusual	0.90468985
up	1.2794247
us	1.5632159
use	1.3566749
venus	6.2337766
very	0.5945349
view	2.2039728
viewing	2.2039728
views	2.2039728
votaries	2.2039728
watched	1.3566749
way	1.1053605
we	1.0297645
wedge	3.8173928
well	1.3566749
went	2.4716613
were	1.4081151
west	1.918628
western	2.932617
what	0.73763573
when	0.66379184
where	1.0297645
whether	1.6931472
which	1.9516007
who	0.62032723
whom	1.6931472
william	2.2039728
witnesses	2.924509
wonder	2.2039728
word	1.6931472
worshippers	2.2039728
would	1.4752715
yards	1.3566749
yds	2.2039728
years	0.73763573
you	0.8407993
```

```
Tg?N???]???k????j???(?b????W??J??_\n??:??I???r?x??:?????e{a??2?W??????????\ZS)?\Z
/\\)?Zw.?-?,?!"\ZC0???j???/.X?y??:?????*??_V`R?????y??????????', 648603518432156
8859)
ENCRYPTED RESULTS:
+
|
+
+
================================================
QUERY: insert into one values('suggests',1693)
NEW QUERY: insert into `test`.`table_VQNGKCVMMT` values ('?\'??p|????????Q*iM?$M
????????l???[?\rL?u??m??P??g', 6670507555341001777, 4166883742714506329, 1082795
2405880536355, 8745816773532767578, '?5@??U}e??????????DB??B????????5?????9/?#???
3?/3F?"n?????!?L?O??@\Z?y1?????(@?ZV??? ?\Z????L?m?:1\0??%?q????<???55a??????xX?
`Z<n?X???b??2?3???j?-???^?%,??????????P?}???#*???P?????:|?t|???@iy???J?,???Cz?l?
??????E?C?#??n????u_????yOS???):??fx>??p?#?[VG??9"????W?', 4291769050706145242)
ENCRYPTED RESULTS:
+
|
+
+
================================================
QUERY: insert into one values('surface',1693)
```

Fig 5.2.8 Inserting into terms and corresponding tfidf values into CryptDB

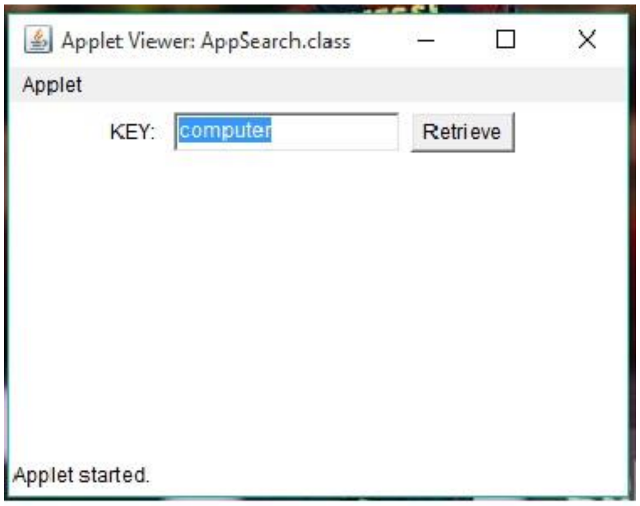
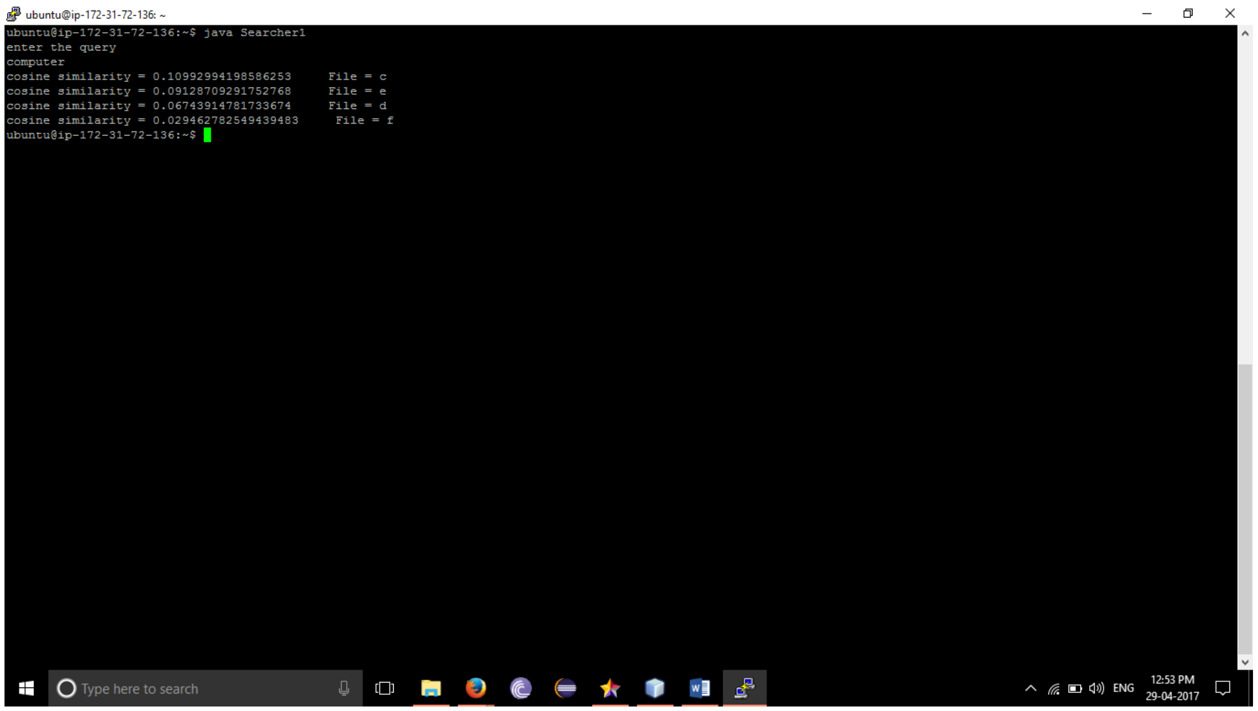


Fig 5.2.9 Reading the query from the user



```
ubuntu@ip-172-31-72-136:~$ java Searcher1
enter the query
computer
cosine similarity = 0.10992994198586253     File = c
cosine similarity = 0.09128709291752768     File = e
cosine similarity = 0.06743914781733674     File = d
cosine similarity = 0.029462782549439483     File = f
ubuntu@ip-172-31-72-136:~$
```

Fig 5.2.10 Displaying the documents after ranking them

6. CONCLUSION

In the face of snooping Database Administrators (DBAs) and compromise from attackers, confidentiality in Database Management Systems (DBMS) should still remain. CryptDB is a system which acts as a proxy to secure the communication between the database server and the applications server. In this project, an application is built over cryptdb that receives query from the user and returns the most relevant documents to the user. Vector based model is employed for information retrieval. In the first step, all the keywords in the document collection and corresponding tf-idf values are send to cryptdb. CryptDB receives them from the application server secures them and sends them to the database server. When the user gives query, application will invoke the CryptDB for tf-idf values. CryptDB will receive the encrypted data from the database decrypts it and sends it to the application server. From these tf-idf values document vectors are created. Then using cosine similarity, the relevant documents are retrieved and ranked and returned to the user. Thus, the entire retrieval process is secured.

6.1 Future Enhancements

- ❖ It can be extended to searching of multimedia files.
- ❖ It can be extended for searching different files like .doc, .pdf.

7. REFERENCES

[1] Raluca Ada Popa, Catherine M. S. Redfield, Nickolai Zeldovich, and Hari Balakrishnan. CryptDB: Protecting Confidentiality with Encrypted Query Processing. In Proceedings of the 23rd ACM Symposium on Operating Systems Principles (SOSP), Cascais, Portugal, October 2011. (This is the main paper describing CryptDB.)

[2] Raluca Ada Popa, Frank H. Li, and Nickolai Zeldovich. An Ideal-Security Protocol for Order-Preserving Encoding. In Proceedings of the 34th IEEE Symposium on Security and Privacy (IEEE S&P/Oakland), San Francisco, CA, May 2013. (This paper constructs the encryption scheme that computes order queries in CryptDB.)

[3] Stephen Tu, M. Frans Kaashoek, Samuel Madden, and Nickolai Zeldovich. Processing Analytical Queries over Encrypted Data. In Proceedings of the 39th International Conference on Very Large Data Bases (VLDB), Riva del Garda, Italy, August 2013. (This paper extends CryptDB's basic design to complex analytical queries and large data sets.)

[4] Raluca Ada Popa and Nickolai Zeldovich. Cryptographic treatment of CryptDB's Adjustable Join. Technical Report MIT-CSAIL-TR-2012-006, Computer Science and Artificial Intelligence Laboratory, Cambridge, MA, March 2012. (A formal description and analysis of CryptDB's adjustable join cryptographic scheme.)

[5] Carlo Curino, Evan P. C. Jones, Raluca Ada Popa, Nirmesh Malviya, Eugene Wu, Sam Madden, Hari Balakrishnan, and Nickolai Zeldovich. Relational Cloud: A Database-as-a-Service for the Cloud. In Proceedings of the 5th Biennial Conference on Innovative Data Systems Research (CIDR 2011), Pacific Grove, CA, January 2011.(A paper describing how CryptDB can help with hosting databases in the cloud.)

[6] Alexandra Boldyreva, Nathan Chenette, and Adam O'Neill. Order-preserving encryption revisited: improved security analysis and alternative solutions.

[7] Seny Kamara. Attacking encrypted database systems, blog post, Outsourced bits, snapshot as of Sept 7, 2015.

[8] Vladimir Kolesnikov and Abdullatif Shikfa. On the limits of privacy provided by order-preserving encryption. Bell Labs Technical Journal, 17(3):135–146, 2012.

[9] Muhammad Naveed, Seny Kamara, and Charles V. Wright. Inference attacks on property-preserving encrypted databases. 2015.

[10] MySQL AB, "MySQL performance benchmarks," A MySQL Technical White Paper, 2005. [Online]. Available: http://www.jonahharris.com/osdb/mysql/mysql-performance-whitepaper.pdf

[11] Google, "Encrypted bigquery client," https://github.com/google/encrypted-bigquery-client, 2015.

[12] P. Grofig, M. Haerterich, I. Hang, F. Kerschbaum, M. Kohler, A. Schaad, A. Schroepfer, and W. Tighzert, "Experiences and observations on the industrial implementation of a system to search over out-sourced encrypted data." in Sicherheit, 2014, pp. 115–125.

[13] I. H. Akin and B. Sunar, "On the difficulty of securing web applications using cryptdb," in Big DataB and Cloud Computing (BdCloud), 2014 IEEE Fourth International Conference on. IEEE, 2014, pp. 745–752.

[14] M. Kantarcioglu and C. Clifton. Security issues in querying encrypted data. Technical Report CSD TR 04-013, Purdue University, Department of Computer Sciences, 2004.

[15] L. Xiong, S. Chitti, and L. Liu. Preserving data privacy for outsourcing data aggregation services. Technical report, Emory University, Department of Mathematics and Computer Science, 2007.

[16] Savoy, J. (1997) Statistical inference in retrieval effectiveness evaluation, Information Processing & Management, 33(4):495-512.

[17] Chen, H., Tobun, D.N., Martinez, J., & Schatz, B. (1997). A Concept Space Approach to Addressing the Vocabulary Problem in Scientific Information Retrieval: An Experiment on the Worm Community System. Journal of the American Society for Information Science.

[18] Hofmann, T. (1999). Probabilistic Latent Semantic Indexing. In Proceedings of the 22st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR’99) (pp. 50-57). ACM.

[19]Raghavan, V.V., & Wong, S.K.M. (1986). A Critical Analysis of Vector Space Model for Information Retrieval. Journal of the American Society for Information Science, 37, 279- 87.

[20]Gotlieb, S.D.J, & Avinash, S. (1968). Semantic clustering of index terms. Journal of the ACM, 15(4), 493-513.

8. APPENDICES

8.1. DocIndexer.java

```
import java.io.*;
import java.sql.SQLException;
import java.util.*;
import org.apache.lucene.analysis.standard.StandardAnalyzer;
import org.apache.lucene.document.Field;
import org.apache.lucene.index.*;
import org.apache.lucene.queryParser.ParseException;
import org.apache.lucene.search.*;
import org.apache.lucene.store.NIOFSDirectory;
import org.apache.lucene.store.RAMDirectory;
import org.apache.lucene.util.Version;

public class DocIndexer {

    private String docNames[];
    private String docIDS[];
    private String pathToIndex;
    private String pathToDocumentCollection;
    private String fiboTermList[]; //marked up fibo terms
    private String taxoTermList[]; // marked up taxonomy terms
    private RAMDirectory ramMemDir;
    private String fileNames[];
    private byte files[][];
    private String filesInText[];
    int noOfWordsOfDOc[];
    int noOfSentencesOfDoc[];
    ArrayList<String> ArrLstSentencesOfDoc[];
    String removedTermsOfDOc[][];
```

```
int freqAfterRemovalOfDoc[][];
private int curDocNo;
private final int maxTerms = 1000000;
int indexDocCount = 0;
public static  ArrayList<String> allTerms = new ArrayList<String>();

public DocIndexer(String pathToIndex, String pathToDocumentCollection) {
    this.pathToIndex = pathToIndex;
    this.pathToDocumentCollection = pathToDocumentCollection;
}
   private int wordCount(String line) {
    int numWords = 0;
    int index = 0;
    boolean prevWhiteSpace = true;
    while (index < line.length()) {
        char c = line.charAt(index++);
        boolean currWhiteSpace = Character.isWhitespace(c);
     if (prevWhiteSpace && !currWhiteSpace) {
            numWords++;
        }
        prevWhiteSpace = currWhiteSpace;
    }
    return numWords;
}

public static String fileReader(String filename) throws IOException {


    String filetext = null;
    BufferedReader reader = null;
    File inFile = new File(filename);
    //READING FROM USERS FILE
    reader = new BufferedReader(new FileReader(inFile));
    String line = null;
```

```
        int numLine = 0;
        while ((line = reader.readLine()) != null) {
                filetext = filetext + " " + line;
        }
        reader.close();
        return filetext;
    }

    public void indexDocs() throws IOException {
        File folder = new File(pathToDocumentCollection);
        File[] listOfFiles = folder.listFiles();
        int noOfFiles = listOfFiles.length;
        System.out.println("Number of files : " + noOfFiles);
        IndexWriter iW;
        try {
            NIOFSDirectory dir = new NIOFSDirectory(new File(pathToIndex));
            iW = new IndexWriter(dir, new IndexWriterConfig(Version.LUCENE_34, new
StandardAnalyzer(Version.LUCENE_34)));
            for (int i = 0; i < noOfFiles; i++) {
                if (listOfFiles[i].isFile()) {
                    String docName = listOfFiles[i].getName();
                    System.out.println("doc name: " + docName + " length - " +
listOfFiles[i].length());
                    if (listOfFiles[i].length() > 1) {
                        String filesInText = fileReader(pathToDocumentCollection + docName);
                        System.out.println("Added to index : " + docName);
                        StringReader strRdElt = new
StringReader(filesInText.replaceAll("\\d+(?:[.,]\\d+)*\\s*", ""));
                        StringReader docId = new StringReader(docName.substring(0,
docName.length() - 4));
                        org.apache.lucene.document.Document doc = new
org.apache.lucene.document.Document();
                        doc.add(new Field("doccontent", strRdElt, Field.TermVector.YES));
                        doc.add(new Field("docid", docId, Field.TermVector.YES));
                        iW.addDocument(doc);
                        indexDocCount++;
                    }
                }
            }

            System.out.println("no of documents added to index : " + indexDocCount);

            iW.close();
        } catch (CorruptIndexException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    public HashMap<Integer, HashMap> tfIdfScore(int numberOfDocs) throws
CorruptIndexException, ParseException, SQLException {
        int noOfDocs = indexDocCount;



        HashMap<Integer, HashMap> scoreMap = new HashMap<Integer, HashMap>();
        //HashMap<Integer, float[]> scoreMap = new HashMap<Integer, float[]>();

        try {

            IndexReader re = IndexReader.open(NIOFSDirectory.open(new
File(pathToIndex)), true);

            int i = 0;
            Cryptcon con = new Cryptcon();
            for (int k = 0; k < numberOfDocs; k++) {
                int freq[];
```

```
TermFreqVector termsFreq;
TermFreqVector termsFreqDocId;
HashMap<String, Float> wordMap = new HashMap<String, Float>();
String terms[];
ArrayList<Integer> frequency = new ArrayList();
float score[] = null;

termsFreq = re.getTermFreqVector(k, "doccontent");
termsFreqDocId = re.getTermFreqVector(k, "docid");
System.out.println(termsFreqDocId.getTerms()[0]);
String name = termsFreqDocId.getTerms()[0];
freq = termsFreq.getTermFrequencies();

terms = termsFreq.getTerms();
for (String ter : terms) {
    if (!allTerms.contains(ter)) {
        allTerms.add(ter);
    }
}
int noOfTerms = terms.length;
score = new float[noOfTerms];
DefaultSimilarity simi = new DefaultSimilarity();
con.create(name);

for (i = 0; i < noOfTerms; i++) {
    int noofDocsContainTerm = re.docFreq(new Term("doccontent", terms[i]));

    float tf = simi.tf(freq[i]);
    float idf = simi.idf(noofDocsContainTerm, noOfDocs);
    float tfidf = tf * idf * 1000;
     frequency.add((int)tfidf);
    System.out.println(terms[i] + "     " + tf * idf);
    wordMap.put(terms[i], (tf * idf));
```

```
            }
            con.insert(name,terms,frequency);
            //scoreMap.put(aInt, wordMap);
        }
        Collections.sort(allTerms);
        con.close();
    } catch (IOException e) {
        e.printStackTrace();
    }
    return scoreMap;
}

public HashMap<Integer, HashMap> getTFIDF() throws IOException,
CorruptIndexException, ParseException, ClassNotFoundException, SQLException {
    int noOfDocs = indexDocCount;
    float tfIdfScore[][] = new float[noOfDocs][];
    HashMap<Integer, HashMap> scoreMap = new HashMap<Integer, HashMap>();

    scoreMap = tfIdfScore(noOfDocs);
    System.out.println(scoreMap);
    return scoreMap;
}

public static void main(String args[]) throws IOException, CorruptIndexException,
ParseException, ClassNotFoundException, SQLException {
    DocIndexer dc = new DocIndexer("/home/ubuntu/doc4/", "/home/ubuntu/doc3/");
    dc.indexDocs();
    dc.getTFIDF();
}
}
```

8.2. Cryptcon.java

```
import java.io.*;
import static java.lang.Character.compare;
import java.sql.*;
import java.util.ArrayList;

public class Cryptcon {
    public class Select_new {
        ArrayList<String> term=new ArrayList<String>();
        ArrayList<Integer> tfidf=new ArrayList<Integer>();
    }

   public Statement stmt;
   Connection con;
   public Cryptcon() {
       try {
           Class.forName("com.mysql.jdbc.Driver");
           con = DriverManager.getConnection("jdbc:mysql://54.208.233.38:3307/test",
"root", "letmein");
           stmt = con.createStatement();
       } catch (Exception e) {
           System.out.println(e.getMessage());
       }
   }

   public void create(String name) throws SQLException {
       System.out.println("create table " + name + "(term varchar(20),tfidf int)");
       stmt.execute("create table " + name + "(term varchar(20),tfidf int)");
   }
```

```
    public void insert(String name, String term[], ArrayList<Integer> a) throws
SQLException {
        // System.out.println("insert into "+name+" values('"+term+"','"+a+"')");
        for (int i = 0; i < term.length; i++) {
    stmt.executeUpdate("insert into " + name + " values('" + term[i] + "'," + a.get(i) + ")");
        }
    }

    public void createallterm() throws SQLException {
        stmt.execute("create table allterms(term varchar(20))");
    }

    public void insertallTerm(String term) throws SQLException {
        stmt.executeUpdate("insert into allterms values('" + term + "')");
    }

    public ArrayList<String> select(String name) throws SQLException {
        ArrayList<String> ip = new ArrayList();
        ResultSet rs = stmt.executeQuery("select *from " + name);
        while (rs.next()) {
            ip.add(rs.getString(1));
        }
        return ip;
    }

    public Select_new selectVector(String name) throws SQLException {
        Select_new termtf=new Select_new();
        ResultSet rs = stmt.executeQuery("select *from " + name);
        while(rs.next()){
            termtf.term.add(rs.getString(1));
            termtf.tfidf.add(rs.getInt(2));
        }
          return termtf;
    }

    public void close() throws SQLException {
        con.close();
    }
}
```

8.3. Searcher.java

```
import java.io.BufferedReader;
import java.io.File;
import java.io.IOException;
import java.io.InputStreamReader;
import java.sql.SQLException;
import java.util.ArrayList;
import java.util.Collections;
import java.util.StringTokenizer;
import static java.lang.Character.compare;
import java.util.Comparator;
import java.util.HashMap;
import java.util.Map;
import java.util.TreeMap;

public class Searcher  {

    public Searcher() {
        this.comp = new Comparator<Double>() {
            @Override
            public int compare(Double o1, Double o2) {
                return o1.compareTo(o2);
            }
        };
    }
   public static Map<Double,String> nonsort=new HashMap<Double,String>();
    public static File folder = new File("/home/ubuntu/doc3/");
    public static File[] listOfFiles = folder.listFiles();
    public static int noOfFiles = listOfFiles.length;
    public static ArrayList<String> filename = new ArrayList<String>();
    public static ArrayList<String> all = new ArrayList<String>();
    public static ArrayList<Docvector> document = new ArrayList<Docvector>();
    public static ArrayList<String> query = new ArrayList<String>();
    public static ArrayList<sorting> selement=new ArrayList<sorting>();
    public static Comparator comp;
    private static void getQueryVector() {
        Docvector d = new Docvector();
        for (String s : all) {
            if (query.contains(s)) {
                d.vector.add(1);
            } else {
                d.vector.add(0);
            }
        }
        document.add(d);
    }

    private static void getTopDocuments() {
        CosineSimilarity con = new CosineSimilarity();
        int size = document.size();
        int j=0;
        for (int i = 0; i < size - 1; i++) {
            double d = con.cosineSimilarity(document.get(i).vector, document.get(size -
1).vector);
            if(d!=0.0){
                nonsort.put(d,document.get(i).name);
        }
    }
            TreeMap<Double,String> sorted=new TreeMap<>(comp);
            sorted.putAll(nonsort);
            Map<Double,String> decsort=sorted.descendingMap();
            printMap(decsort);
    }
     public static <K, V> void printMap(Map<K, V> map) {
        for (Map.Entry<K, V> entry : map.entrySet()) {
            System.out.println("cosine similarity = " + entry.getKey()
                                        + "     File = " + entry.getValue());
        }
    }


    public static class Docvector {

        String name;
        ArrayList<Integer> vector = new ArrayList();
    }

    public static void main(String args[]) throws SQLException, IOException {
        for (int i = 0; i < noOfFiles; i++) {
            String name = listOfFiles[i].getName();
            int length = name.length();
            filename.add(name.substring(0, length - 4));
        }
        addallterm();
        all.remove("null");
        Collections.sort(all);
        addvecname();
       getQuery();
        getQueryVector();
       getTopDocuments();
```

```
}

 private static void addvecname() throws SQLException {
     Cryptcon con = new Cryptcon();
     for (int i = 0; i < noOfFiles; i++) {
         Docvector v = new Docvector();
         v.name = filename.get(i);
             Cryptcon.Select_new tfidf = con.selectVector(filename.get(i));
            for(String t:all){
                char c=t.charAt(0);
                for(int j=0;j<tfidf.term.size();j++){
                    int x=compare(c,tfidf.term.get(j).charAt(0));
                      if(x < 0) {
                         v.vector.add(0);
                         break;
                             }
                      if(t.equals(tfidf.term.get(j))){
                          v.vector.add(tfidf.tfidf.get(j));
                          break;
                      }
                }
            }
          document.add(v);
      }
 }


 private static void getQuery() throws IOException {
     BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
     System.out.println("enter the query");
     String s = br.readLine();
     StringTokenizer str = new StringTokenizer(s, " ");
     while (str.hasMoreTokens()) {
            query.add(str.nextToken());
        }
    }

    private static void addallterm() throws SQLException {
        Cryptcon con = new Cryptcon();
        for (int i = 0; i < noOfFiles; i++) {
            ArrayList<String> term = new ArrayList();
            term = con.select(filename.get(i));
            //System.out.println("testing "+filename.get(i));
            for (String t : term) {
                if (!all.contains(t)) {
                    all.add(t);
                }
            }
        }
    }
}
```

8.4. CosineSimilarity.java

```
import java.util.ArrayList;
public class CosineSimilarity {

public double cosineSimilarity(ArrayList<Integer> docVector1, ArrayList<Integer>
docVector2) {
        double dotProduct = 0.0;
        double magnitude1 = 0.0;
        double magnitude2 = 0.0;
        double cosineSimilarity = 0.0;

        for (int i = 0; i < docVector1.size(); i++) //docVector1 and docVector2 must be of
same length
```

```
        {
            dotProduct += docVector1.get(i)/1000 * docVector2.get(i);  //a.b
            magnitude1 += Math.pow((docVector1.get(i)/1000), 2);  //(a^2)
            magnitude2 += Math.pow((docVector2.get(i)), 2); //(b^2)
        }

        magnitude1 = Math.sqrt(magnitude1);//sqrt(a^2)
        magnitude2 = Math.sqrt(magnitude2);//sqrt(b^2)

        if (magnitude1 != 0.0 && magnitude2 != 0.0) {
            cosineSimilarity = dotProduct / (magnitude1 * magnitude2);
        } else {
            return 0.0;
        }
        return cosineSimilarity;
    }
}
```