**Mona Secure Multi-Owner Data Sharing for Dynamic Groups in the Cloud**

**ABSTRACT:**

With the character of low maintenance, cloud computing provides an economical and efficient solution for sharing group resource among cloud users. Unfortunately, sharing data in a multi-owner manner while preserving data and identity privacy from an untrusted cloud is still a challenging issue, due to the frequent change of the membership. In this paper, we propose a secure multi owner data sharing scheme, named Mona, for dynamic groups in the cloud. By leveraging group signature and dynamic broadcast encryption techniques, any cloud user can anonymously share data with others. Meanwhile, the storage overhead and encryption computation cost of our scheme are independent with the number of revoked users. In addition, we analyze the security of our scheme with rigorous proofs, and demonstrate the efficiency of our scheme in experiments.

**KEYWORDS:** Cloud computing, data sharing, privacy-preserving, access control, dynamic groups

**LIST OF CONTENTS**

Page No

List of Figures viii

List of Tables ix

1. Introduction

1.1 Purpose

1.2 Scope

1.3 Motivation

1.3.1 Definitions

1.3.2 Abbreviations

1.3.3 Model Diagrams

1.4 Overview

2. Literature Survey

2.1 Introduction

2.2 History

2.3 Purpose

2.4 Requirements

2.5 Technology Used

2.6 Research Methodologies

3. Fundamental Concepts on (Domain)

3.1 Domain Fundamentals & Description

3.1 Existing concepts of fundamentals

3.2 Existing System Algorithms

3.3 Proposed System Fundamentals concepts

3.4 Proposed Algorithms

3.5 Performance analysis in between of existing system and proposed system

4. System Analysis

4.1 Existing System

4.1.1 Drawbacks

4.2 Problem statement

4.3 Proposed System

4.3.1 Advantages

4.4 Modules Description

4.5 Feasibility Study

4.5.1 Economic Feasibility

4.5.2 Operational Feasibility

4.5.3 Technical Feasibility

5. System Requirements Specification

5.1 Introduction

5.2 Purpose

5.3 Functional Requirements

5.4 Non Functional Requirements

5.5 Hardware Requirements

5.6 Software Requirements

6. System Design

6.1 System Specifications

6.2 System Components

6.3 DFD’s

6.4 UML Diagrams

6.5 Data Dictionaries and ER Diagram

7. Implementation

7.1 Technology Description

7.2 Code Snippets (Logics) & Analysis

8. System Testing

8.1 Testing Methodologies

8.2 Test cases

8.3 Result Analysis

9. Conclusion and Future Enhancements

9.1 Conclusion

9.2 Scope for Future Enhancements

10. References

Appendix:

1. Sample code
2. screenshots

**Introduction:**

CLOUD computing is recognized as an alternative to traditional information technology [1] due to its intrinsic resource-sharing and low-maintenance characteristics. In cloud computing, the cloud service providers (CSPs), such as Amazon, are able to deliver various services to cloud users with the help of powerful data centers. By migrating the local data management systems into cloud servers, users can enjoy high-quality services and save significant investments on their local infrastructures. One of the most fundamental services offered by cloud providers is data storage. Let us consider a practical data application. A company allows its staffs in the same group or department to store and share files in the cloud. By utilizing the cloud, the staffs can be completely released from the troublesome local data storage and maintenance. However, it also poses a significant risk to the confidentiality of those stored files. Specifically, the cloud servers

managed by cloud providers are not fully trusted by users while the data files stored in the cloud may be sensitive and confidential, such as business plans. To preserve data privacy, a basic solution is to encrypt data files, and then upload the encrypted data into the cloud [2]. Unfortunately, designing an efficient and secure data sharing scheme for groups in the cloud is not an easy task due to the following challenging issues.

First, identity privacy is one of the most significant obstacles for the wide deployment of cloud computing.

Without the guarantee of identity privacy, users may be unwilling to join in cloud computing systems because their real identities could be easily disclosed to cloud providers and attackers. On the other hand, unconditional identity privacy may incur the abuse of privacy. For example, a misbehaved staff can deceive others in the company by sharing false files without being traceable. Therefore, traceability, which enables the group manager (e.g., a company manager) to reveal the real identity of a user, is also highly desirable. Second, it is highly recommended that any member in a group should be able to fully enjoy the data storing and sharing services provided by the cloud, which is defined as the multiple-owner manner. Compared with the single-owner manner [3], where only the group manager can store and modify data in the cloud, the multiple-owner manner is more flexible in practical applications. More concretely, each user in the group is able to not only read data, but also modify his/ her part of data in the entire data file shared by the company. Last but not least, groups are normally dynamic in practice, e.g., new staff participation and current employee revocation in a company. The changes of membership make secure data sharing extremely difficult. On one hand, theanonymous system challenges new granted users to learn the content of data files stored before their participation, because it is impossible for new granted users to contact with anonymous data owners, and obtain the corresponding

decryption keys. On the other hand, an efficient membership revocation mechanism without updating the secret keys of the remaining users is also desired to minimize the complexity of key management.

Several security schemes for data sharing on untrusted servers have been proposed [4], [5], [6]. In these approaches, data owners store the encrypted data files in untrusted storage and distribute the corresponding decryption keys only to authorized users. Thus, unauthorized users as well as storage servers cannot learn the content of the data files because they have no knowledge of the decryption keys.

**1.1 Purpose**

We propose a secure multi-owner data sharing scheme. It implies that any user in the group can securely share data with others by the untrusted cloud.

Our proposed scheme is able to support dynamic groups efficiently. Specifically, new granted users can directly decrypt data files uploaded before their participation without contacting with data owners. User revocation can be easily achieved through a novel revocation list without updating the secret keys of the remaining users. The size and computation overhead of encryption are constant and independent with the number of revoked users.

We provide secure and privacy-preserving access control to users, which guarantees any member in a group to anonymously utilize the cloud resource. Moreover, the real identities of data owners can be revealed by the group manager when disputes occur.

We provide rigorous security analysis, and perform extensive simulations to demonstrate the efficiency of our scheme in terms of storage and computation overhead.

**1.2 Scope**

**1.3 Motivation**

**1.3.1 Definitions**

The main contributions of this paper include: We propose a secure multi-owner data sharing scheme. It implies that any user in the group can securely share data with others by the untrusted

Cloud. Our proposed scheme is able to support dynamic groups efficiently. Specifically, new granted users can directly decrypt data files uploaded before their participation without contacting with data owners. User revocation can be easily achieved through a novel revocation list without updating the secret keys of the remaining users. The size and computation overhead of encryption are constant and independent with the number of revoked users. We provide secure and privacy-preserving access control to users, which guarantees any member in a group to anonymously utilize the cloud resource. Moreover, the real identities of data owners can be revealed by the group manager when disputes occur. We provide rigorous security analysis, and perform extensive simulations to demonstrate the efficiency of our scheme in terms of storage and computation overhead.

**1.3.2 Abbreviations**

**1.3.3 Model Diagrams**

**Architecture:**

**1.4 Overview**

**Section1.** In this section we have discussion Introduction, Purpose, Scope and Motivation

**Section2.** In this section we will discussion Literature survey and Research Methodologies.

**Section3.** Inthis section we will discussion,Fundamental Concepts on Data Mining, Existing Techniques, and Proposed Techniques and Algorithm

**Section4.** Inthis section we will discussion System Analysis, Module Description and Feasibility Study.

**Section5.** In this section we will discussion System Requirements Specification.

**Section6.** In this section we will discussion System Design (UML, DFD, and ER Diagram).

**Section7.** In this section we will discussion Implementation and Technology Description.

**Section8.** In this section we will discussion System Testing and Result Analysis.

**Section9.** In this section we will discussion Conclusion and Future Enhancement.

**Section10.** In this section we will discussion References.

**Litarature SERVEY:**

We consider a sensor network that is not fully trusted and ask the question how we preserve privacy for the collected data and how we verify the data reply from the network. We explore the problem in the context of a network augmented with storage nodes and target at range query. We use bucketing scheme to mix the data for a range, use message encryption for data integrity, and employ encoding numbers to prevent the storage nodes from dropping data.

I. INTRODUCTION

We believe that pervasive computing systems, touching upon every aspect of our life, will be partially supported by the sensor network infrastructure, which is involved in two processes: monitoring the environment surrounding us (also including us), and providing information for us to analyze and respond. Both processes are exposed to potential risks for information security and privacy prohibiting the realistic sensor network deployment. On one hand, a sensor network may leak information about people to an unauthorized party, which leads to a privacy breaching. On the other hand, it may also lie about the collected data to a valid query making the network dysfunctional. In deploying such a realistic sensor network, a fundamental question is how much we should trust a sensor network and how we prevent, or at least, to detect the misbehavior of the sensor network. Unfortunately, little research work has targeted to solve the problem. This paper tries to address the problem in a setting of network enhanced

by some nodes with large storage space and considers a powerful and typical sensor network operation: range query.

The network setting, we believe, will be a natural enhancement to the future sensor network. Range query is powerful enough to cover many interesting types of queries including location based queries. Thus, our model is generalized enough for us to investigate the trust problem in a practical and also meaningful environment.

We envision that future sensor network shall be augmented by sparsely deployed special nodes for data storage. Those storage nodes differ from the regular sensors with a larger storage space (e.g., with more enriched flash memory). Sensor network generates a large amount of data, and, many times, the collected data has to be archived for future retrieval. Data can be stored in the sensor nodes or sent back to the base station,

each of which has its limitation. To store data on the sensor nodes is prohibitive due to the limited storage space on each sensor node and the difficulty in collecting all the data to a central repository. Transmitting all the data to the base station, on the other hand, has to address the limited transmission rate that is especially throttled by the funnel effect around the base station and attenuated per node transmission bandwidth. The introduction of the storage nodes helps to alleviate the transmission bandwidth problem by distributing the local data transmission to the storage node.

This hierarchical structure has been instantiated by the recently popular stargate device [1] and the memory-enhanced sensor nodes by UC Riverside [2]. Those special powerful nodes take advantage of their high transmission capability and storage and even computational capability to alleviate the cursed bandwidth limitation, and also provide auxiliary support for surrounding vulnerable sensors for data back-up. The introduction of the storage node is also spurred by the recent concept of “data-centric storage” [3]. Data-centric storage deterministically conducts a mapping between the name of a data (N) to the address associated with a specific

node (A). All the data with name N generated by the network are accumulated to the node A and all queries about N go to A too. In this way, network-wide search for data query is avoided dramatically reducing communication cost in many scenarios. For example, a sensor network deployed for plant monitoring

may forward queries about the humidity to a storage sensor directly, and the temperature to another storage sensor instead of querying the entire network. Storage nodes have to take care of regular real-time sensor

network operations, for example, data query, so that network user may monitor the sensor field and respond with the environmental change in a timely fashion. This natural requirement implies that the storage node has to gain some understanding about the stored data for an energy-efficient data reply by avoiding sending all the collected data back. The practice would not be a problem if the storage nodes are trusted as most of the sensor network research assumes. It is not valid, however, if the storage nodes are susceptible to compromise

and the disclosure of the information may endanger the crucial assignment for the users in the network. With more sensor network deployed for pervasive computing applications, this issue becomes even more serious if the user information is leaked through the storage nodes, which breaches the privacy requirement.

Generally, an adversary is not able to compromise numerous deployed sensors. The limited number of compromised sensors do not affect the query reply seriously because of redundant sensor deployment and limited coverage of the compromised sensors. The storage nodes, which hold much data collected from many sensors, however, will be the target for compromise and have to be a great concern when privacy related information is collected and query is imposed to the collected data. In this paper, we are particularly interested in the privacy

implication of this network architecture, which we believe will be prevalent in the future sensor network deployment. We focus on data range query, which asks the storage nodes to return the data in a range specified by [a, b] with the sensor ids attached to each data1. We would like the storage nodes

reply the range query without gaining too much information. Since the storage nodes are not trusted, it is very likely that they lie about the collected data or discard some data. To prevent the malicious or non-cooperative storage nodes from cheating or dropping the data is hard, but at least the user is entitled to know whether the data reply is intact. In a word, we require the range query in this network setting to be privacy preserving and verifiable. Even though we use storage node as part of our network structure, the concept and the techniques used in this paper can

also be applied to the regular range query in which aggregating nodes accumulate the results from the sensors in their charge. In storage node case, data has already been transmitted to the storage nodes before the query arrival, while in aggregating node case, the queries will be dissipated to the sensors online and the aggregating nodes will simply accumulate the replies and forward to the base station. To the best of our knowledge, this paper is the first in

dealing with privacy issue in sensor network data query. Even though there are papers on providing secure sensor network operations and verifying sensor operation results, most of them are focused on data aggregation, but not on more complicated range query, which is common and important in many sensor network applications. Our work explores the privacy concerns in sensor network in a very general setting and provides meaningful and interesting results for data reply verification.

We consider a sensor network that is not fully trusted and ask the question how we preserve privacy for the collected data and how we verify the data reply from the network. We explore the problem in the context of a network augmented with storage nodes and target at range query. We use bucketing scheme to mix the data for a range, use message encryption for data integrity, and employ encoding numbers to prevent the storage nodes from dropping data.

I. INTRODUCTION

We believe that pervasive computing systems, touching upon every aspect of our life, will be partially supported by the sensor network infrastructure. This infrastructure will monitor the environment surrounding us (also including us), and provide information for us to analyze and respond. Since it collects information about people, security and privacy become a big concern. Indeed, security and privacy breaching can happen in any link. For example, a sensor network may leak information about people to an unauthorized party; it may also lie about the collected data to a valid query making the network dysfunctional. In deploying such a realistic sensor network, a fundamental question is how much we should trust the sensor network and how we prevent, or at least, detect the misbehavior of the sensor network. Unfortunately, little research work has managed to solve the problem. This paper tries to address the problem in a two-tiered network where some nodes are equipped with much larger storage than regular sensors, which we call *storage nodes*. This network setting, we believe, will be a natural enhancement to the future sensor networks. Under this network architecture, we consider range query, a typical sensor network operation, which is very powerful to cover many interesting types of queries. We feel that our model in this paper is generalized enough to investigate the trust problem in a practical and also meaningful environment.

The inclusion of storage nodes in this two tiered architecture is owing to two considerations. First, transferring the collected data to the base station consumes too much energy and creates communication bottleneck close to the base station [1]. Thus, in-network storage is necessary. Second, provisioning all sensors with large storage is less attractive because querying the network is tantamount to searching all the sensors in the

network, which consumes much energy [1]. In addition, even though the storage becomes quite inexpensive, large storage in This project was supported in part by US National Science Foundationaward CCF-0514985 and CNS-0721443. numerous sensors would still be a hurdle for realistic deployment. The integration of storage node is also substantiated by the belief that the tiered architecture for sensor networks is more practical [2] and the advent of the new storage-enriched hardware [3]–[5].

In this paper, we consider general applications concerned with data range query, which asks storage nodes to return the data in a range specified by [*a, b*].1 Particularly, we are interested in the security issues under this two-tiered model. When deployed in a hostile environment, storage nodes could easily become the target for compromise due to their important rolein this accessing model. Two threats arise when storage nodes are compromised. First, the compromised storage nodes may disclose the data stored on them to the adversary and breach data privacy. Resolving this threat is challenging because storage nodes have to gain information about the collected data to respond to a range query, which is in conflict with the privacy requirement. Second, the compromised storage nodes may lie about the collected data and send wrong information as the reply. This attack is very hard to prevent, because the compromised storage node may be fully controlled by the adversary. In this paper, we propose solutions to solving these two problems. For the first threat, our scheme strikes a balance in how much information to release so that data query can be performed while privacy will not be much harmed. For the second threat, we propose a passive solution to enable the user to verify whether the reply is intact. Our major contributions are: (1) To the best of our knowledge, this paper is the first that considers the privacy issue when processing range query in sensor networks. Our work explores the privacy concerns in sensor networks in a very general setting and provides meaningful and interesting results for data reply verification. (2) We propose a privacy-preserving

storage scheme, in which only coarse information is disclosed to storage nodes while data can still be processed upon the range query. (3) We introduce an encoding scheme, which allows the sink to verify the reply of a range query with small lextra overheads incurred. (4) Finally, we evaluate our solutions by comprehensive simulation based on synthetic and real data sets, and our results show that the proposed schemes achieve the privacy and security goals efficiently. The rest of this paper is organized as follows. In Section II,

1Range query is a very powerful type of query, e.g., event detection can be quantified as querying the range of the monitored variables associated with the event.

Algorithm 1 Optimal Solution (*F, V ARp,ENp, α, δ*)

1: for *i* = *vmin* to *vmax* do

2: for *j* = *i* to *vmax* do

3: Calculate ¯*E*[*i, j*] and *PT*[*i, j*] by Eq.(1)

4: Calculate variance and entropy by Eq.(2) and Eq.(3)

5: if variance *> VARp* and entropy *> ENp* then

6: *valid*[*i, j*] = *true*, *D*[*i, j*] = *EncodingLength*([*i, j*])

7: *COST*[*i, j*] = Eq.(5)+Eq.(4)

8: for *w* = 1 to *vmax − vmin* + 1 do

9: for *i* = 1 to *vmax − w* do

10: if *valid*[*i, i* + *w*] then

11: *M*[*i, i* + *w*] = *COST*[*i, j*]

12: for *j* = 1 to *w −* 1 do

13: if *valid*[*i, i* + *j*] then

14: *cost* = *COST*[*i, i* + *j*] +*M*[*i* + *j* + 1*, i* + *w*]

15: if *cost < M*[*i, i* + *w*] then

16: *M*[*i, i* + *w*] = *cost, P*[*i, i* + *w*] = *j*

17: return *D*, *M* and *P*

In 1998, Blaze, Bleumer, and Strauss (BBS) proposed an application called *atomic proxy re-encryption*, in which a semi-trusted proxy converts a ciphertext for Alice into a ciphertext for Bob *without* seeing the underlying plaintext. We predict that fast and secure reencryption will become increasingly popular as a method for managing encrypted file systems. Although efficiently computable, the wide-spread adoption of BBS re-encryption has been hindered by considerable security risks. Following recent work of Dodis and Ivan, we present new re-encryption schemes that realize a stronger notion of security, and we demonstrate the usefulness of proxy re-encryption as a method of adding access control to a secure file system.

Performance measurements of our experimental file system demonstrate that proxy re-encryption can work effectively in practice.

1 Introduction

Proxy re-encryption allows a proxy to transform a ciphertext computed under Alice’s public key into one that can be opened by Bob’s secret key. There are many useful applications of this primitive. For instance, Alice might wish to temporarily forward encrypted email to her colleague Bob, without giving him her secret key. In this case, Alice the delegator could designate a proxy to re-encrypt her incoming mail into a format that Bob the delegatee can decrypt using his own secret key. Alice could simply provide her secret key to the proxy, but this requires an unrealistic level of trust in the proxy. We present several efficient proxy re-encryption schemes that offer security improvements over earlier approaches. The primary advantage of our schemes is that they are unidirectional (i.e., Alice can delegate to Bob without Bob having to delegate to her) and do not require delegators to reveal all of their secret key to anyone – or even interact with the delegatee – in order to allow a proxy to re-encrypt their ciphertexts.

In our schemes, only a limited amount of trust is placed in the proxy. For example, it is not able to decrypt the ciphertexts it re-encrypts, and we prove our schemes secure even when the proxy publishes all the reencryption information it knows. This enables a number of applications that would not be practical if the proxy needed to be fully trusted.

We present a new methodology for realizing Cipher text-Policy Attribute Encryption (CP- ABE) under concrete and no interactive cryptographic assumptions in the standard model. Our solutions allow any encrypt or to specify access control in terms of any access formula over the attributes in the system. In our most efficient system, cipher text size, encryption, and decryption time scales linearly with the complexity of the access formula. The only previous work to achieve these parameters was limited to a proof in the generic group model.

We present three constructions within our framework. Our first system is proven selectively secure under a assumption that we call the decisional Parallel Bilinear Die-Hellman Exponent (PBDHE) assumption which can be viewed as a generalization of the BDHE assumption. Our next two constructions provide performance tradeoffs to achieve provable security respectively under the (weaker) decisional Bilinear-Daffier-Hellman Exponent and decisional Bilinear Diffie-

Hellman assumptions

Public-Key encryption is a powerful mechanism for protecting the confidentiality of stored and

transmitted information. Traditionally, encryption is viewed as a method for a user to share data

to a targeted user or device. While this is useful for applications where the data provider knows

specifically which user he wants to share with, in many applications the provider will want to share data according to some policy based on the receiving user's credentials. Sahai andWaters presented a new vision for encryption where the data provider can express how he wants to share data in the encryption algorithm itself. The data provider will provide a predicate f(\_) describing how he wants to share the data and a user will be ascribed a secret key associated with their credentials X; the user with credentials X can decrypt a cipher text encrypted with predicate f if f(X) = 1. Sahai and Waters [36] presented a particular formulation of this problem that they called Attribute-Based Encryption (ABE), in which a user's credentials is represented by a set of string called \attributes" and the predicate is represented by a formula overthese attributes. Several techniques used by SW were inspired by prior work on Identity-Based Encryption [37, 13, 24, 18, 10]. One drawback of the Sahai-Waters approach is that their initial construction was limited to handling formulas consisting of one threshold gate. In subsequent work, Goyal, Pandey, Sahai, and Waters further clarified the concept of Attribute-Based Encryption. In particular, they proposed two complementary forms of ABE. In

the first, Key-Policy ABE, attributes are used to annotate the ciphertexts and formulas over these attributes are ascribed to users' secret keys. The second type, Ciphertext-Policy ABE, is comple- mentary in that attributes are used to describe the user's credentials and the formulas over these credentials are attached to the ciphertext by the encrypting party. In addition, Goyal et al. [28] provided a construction for Key-Policy ABE that was very expressive in that it allowed the policies (attached to keys) to be expressed by any monotonic formula over encrypted data. The system was proved selectively secure under the Bilinear Di\_e-Hellman assumption. However,

Cloud computing is an emerging computing paradigm in which resources of the computing infrastructure are provided as services over the Internet. As promising as it is, this paradigm also brings forth many new challenges for data security and access control when users outsource sensitive data for sharing on cloud servers, which are not within the same trusted domain as data owners. To keep sensitive user data confidential against untrusted servers, existing solutions usually apply cryptographic methods by disclosing data decryption keys only to authorized users. However, in doing so, these solutions inevitably introduce a heavy computation overhead on the data owner for key distribution and data management when finegrained data access control is desired, and thus do not scale well. The problem of simultaneously achieving fine-grainedness, scalability, and data confidentiality of access control actually still remains unresolved. This paper addresses this challenging open issue by, on one hand, defining and enforcing access policies based on data attributes, and, on the other hand, allowing the data owner to delegate most of the computation tasks involved in finegrained data access control to untrusted cloud servers without disclosing the underlying data contents. We achieve this goal by

exploiting and uniquely combining techniques of attribute-based encryption (ABE), proxy re-encryption, and lazy re-encryption. Our proposed scheme also has salient properties of user access privilege confidentiality and user secret key accountability. Extensive analysis shows that our proposed scheme is highly efficient and provably secure under existing security models.

I. INTRODUCTION

Cloud computing is a promising computing paradigm which recently has drawn extensive attention from both academia and industry. By combining a set of existing and new techniques from research areas such as Service-Oriented Architectures (SOA) and virtualization, cloud computing is regarded as such a computing paradigm in which resources in the computing infrastructure are provided as services over the Internet. Along

with this new paradigm, various business models are developed, which can be described by terminology of “X as a

service (XaaS)” [1] where X could be software, hardware, data storage, and etc. Successful examples are Amazon’s EC2and S3 [2], Google App Engine [3], and Microsoft Azure [4] which provide users with scalable resources in the pay-as-youuse fashion at relatively low prices. For example, Amazon’s S3 data storage service just charges $0.12 to $0.15 per gigabytemonth. As compared to building their own infrastructures, users are able to save their investments significantly by migrating businesses into the cloud. With the increasing development

of cloud computing technologies, it is not hard to imagine that in the near future more and more businesses will be moved into the cloud. As promising as it is, cloud computing is also facing many

challenges that, if not well resolved, may impede its fast growth. Data security, as it exists in many other applications, is among these challenges that would raise great concerns from users when they store sensitive information on cloud servers. These concerns originate from the fact that cloud severs are usually operated by commercial providers which are very likely to be outside of the trusted domain of the users.

Data confidential against cloud servers is hence frequently desired when users outsource data for storage in the cloud. In some practical application systems, data confidentiality is not

only a security/privacy issue, but also of juristic concerns. For example, in healthcare application scenarios use and disclosure of protected health information (PHI) should meet the requirements of Health Insurance Portability and Accountability Act (HIPAA) [5], and keeping user data confidential against the storage servers is not just an option, but a requirement. Furthermore, we observe that there are also cases in which cloud users themselves are content providers. They publish data on cloud servers for sharing and need fine-grained data access control in terms of which user (data consumer) has the access privilege to which types of data. In the healthcare case, for example, a medical center would be the data owner who stores millions of healthcare records in the cloud. It would allow data consumers such as doctors, patients, researchers and etc, to access various types of healthcare records under policies admitted by HIPAA. To enforce these access policies, the data owners on one hand would like to take advantage of the abundant resources that the cloud provides for efficiency and economy; on the other hand, they may want to keep the data contents confidential against cloud servers. As a significant research area for system protection, data access control has been evolving in the past thirty years and various techniques [6]–[9] have been developed to effectively implement fine-grained access control, which allows flexibility

in specifying differential access rights of individual users. Traditional access control architectures usually assume the data owner and the servers storing the data are in the same trusteddomain, where the servers are fully entrusted as an omniscient reference monitor [10] responsible for defining and enforcing access control policies. This assumption however no longer holds in cloud computing since the data owner and cloud servers are very likely to be in two different domains. On one hand, cloud servers are not entitled to access the outsourced

data content for data confidentiality; on the other hand, the data resources are not physically under the full control of

2 the owner. For the purpose of helping the data owner enjoy fine-grained access control of data stored on untrusted cloud servers, a feasible solution would be encrypting data through certain cryptographic primitive(s), and disclosing decryption keys only to authorized users. Unauthorized users, including cloud servers, are not able to decrypt since they do not have the data decryption keys. This general method actually has been widely adopted by existing works [11]–[14] which aim at securing data storage on untrusted servers. One critical issue

with this branch of approaches is how to achieve the desired security goals without introducing a high complexity on key management and data encryption. These existing works, as we will discuss in section V-C, resolve this issue either by introducing a per file access control list (ACL) for fine-grained access control, or by categorizing files into several filegroups for efficiency. As the system scales, however, the complexity of the ACL-based scheme would be proportional to the number of users in the system. The filegroup-based scheme, on

the other hand, is just able to provide coarse-grained data access control. It actually still remains open to simultaneously achieve the goals of fine-grainedness, scalability, and data confidentiality for data access control in cloud computing.

In this paper, we address this open issue and propose a secure and scalable fine-grained data access control scheme

for cloud computing. Our proposed scheme is partially based on our observation that, in practical application scenarios each data file can be associated with a set of attributes which are meaningful in the context of interest. The access structure of each user can thus be defined as a unique logical expression

over these attributes to reflect the scope of data files that the user is allowed to access. As the logical expression can

represent any desired data file set, fine-grainedness of data access control is achieved. To enforce these access structures,we define a public key component for each attribute. Data files

are encrypted using public key components corresponding to their attributes. User secret keys are defined to reflect their access structures so that a user is able to decrypt a ciphertext if and only if the data file attributes satisfy his access structure. Such a design also brings about the efficiency benefit, as compared to previous works, in that, 1) the complexity of encryption is just related the number of attributes associated to the data file, and is independent to the number of users in the system; and 2) data file creation/deletion and new user grant operations just affect current file/user without involving system-wide data file update or re-keying. One extremely challenging issue with this design is the implementation of user revocation, which would inevitably require re-encryption of data files accessible to the leaving user, and may need update of secret keys for all the remaining users. If all these tasks are performed by the data owner himself/herself, it wouldintroduce a heavy computation overhead on him/her and may

also require the data owner to be always online. To resolve this challenging issue, our proposed scheme enables the data owner to delegate tasks of data file re-encryption and user secret key update to cloud servers without disclosing data contents or user access privilege information. We achieve our design goals by exploiting a novel cryptographic primitive, namely key policy attribute-based encryption (KP-ABE) [15], and uniquely combine it with the technique of proxy reencryption (PRE) [16] and lazy re-encryption [11]. Main contributions of this paper can be summarized as follows. 1) To the best of our knowledge, this paper is the first that simultaneously achieves fine-grainedness, scalability and data confidentiality for data access control in cloud computing;

2) Our proposed scheme enables the data owner to delegate most of computation intensive tasks to cloud servers without disclosing data contents or user access privilege information; 3) The proposed scheme is provably secure under the standard security model. In addition, our proposed scheme is able to support user accountability with minor extension.

**3. Fundamental Concepts on (Domain)**

**Domain (Cloud Computing) Fundamentals & Description**

**1.1 Defining Cloud Computing**

Cloud computing refers to both the applications delivered as services over the Internet and the hardware and systems software in the data centers that provide those services. These services have long been referred to as Software as a Service (SaaS). Some terms such as PaaS (Platform as a Service) and IaaS (Infrastructure as a Service) are used by vendors to describe their products, but we avoid these because accepted definitions for them still vary widely. There is no crisp line between “low-level “infrastructure and a higher-level “platform “. We believe both of these are more alike than different, and we do consider them together. Similarly, some related term such as “grid computing,” from the high-performance computing community, suggests protocols to offer storage over long distances and shared computation, however those protocols did not lead to a software environment that grew beyond its own community. The data center hardware and software is what we will call a *cloud*. When a cloud is made available in a pay-as you- go manner to the general public, we call it a *public cloud*; the service being sold is *utility computing*. We use the term *private cloud* to refer to internal data centers of a business or other organization, not made available to the general public, when they are large enough to benefit from the advantages of cloud computing that we discuss here. The cloud computing is the sum of SaaS and utility computing, but does not include medium sized data centers, even if these depend on virtualization for management. People can be users or providers of SaaS, or users or providers of utility computing. We focus on SaaS providers (cloud users) and cloud providers, which have received less attention than SaaS users. Figure 1 makes provider-user relationships clear. There are some case in which the same actor plays multiple roles. For instance, a cloud provider might also host its own customer-facing services on cloud infrastructure.

**1.2 All kinds of clouds**

Major IT companies have spent billions of dollars since the 1990s to shape cloud computing. Like, Sun’s well-known slogan “the network is the computer” was made in 1980s.

Salesforce.com is the website which has been providing on-demand Software as a Service (SaaS) for customers since 1999 to present era. IBM and Microsoft are the first two companies that started to deliver Web services in the early 2000s. Microsoft’s Azure service provides an operating system and a set of developer tools and services. Google’s popular Google Docs software provides Web-based word-processing, spreadsheets and all the Microsoft office applications. Google App Engine allows system developers to run their Python/Java applications on Google’s infrastructure. Sun provides $1 per CPU hour. Amazon is well-known for providing Web services such as EC2 and S3. Yahoo! announced that it would use the Apache Hadoop framework to allow users to work with thousands of nodes and petabytes (1 million gigabytes) of data.



These examples demonstrate that cloud computing providers are offering services on every level, from different hardware (e.g., Amazon and Sun), to the different operating systems (e.g., Google and Microsoft), to software and different services (e.g., Google, Microsoft, and Yahoo!). At present era Cloud-computing providers target a variety of end users, from developers of the software to the general public. For additional information regarding cloud computing models, the University of California (UC) Berkeley’s report provides a good comparison of these models by Amazon, Microsoft, and Google. As cloud computing providers prices are low and IT advancements remove technology barriers—such as virtualization, simulation, network bandwidth — cloud computing has moved into the mainstream of technology . Gartner stated, “Organizations are switching from company owner hardware and software to per-use service-based models.” For example, the U.S. government website (http://www.usa .gov/) will soon begin using cloud computing. The *New York Times* used Amazon’s EC2 and S3 services and used Hadoop application to provide open access for the public domain articles from 1851 to 1922. The *Times* loaded 4 TB of raw TIFF images on web and their derivative 11 million PDFs into Amazon’s S3 in twenty-four hours at very less cost. This project is very similar to digital library projects run by academic libraries. Few years ago OCLC announced its movement of library management services to the Web It is clear that OCLC is going to deliver a Web-based integrated library system (ILS) on web for enhancing the technology to provide a new way of running an ILS. Dura Space, a joint organization by Fedora Commons and D Space Foundation, announced that they would be taking advantage of cloud storage and cloud computing.

**2. Delivery Models of Cloud Computing**

The NIST definition of cloud computing defines three delivery models:



**2.1. Software as a Service (SaaS)**

The consumer uses an application, but does not control the operating system, hardware or network infrastructure on which it’s running. The SaaS model shown in the diagram admits that

the provider manages the entire suite of applications delivered to end-users. SaaS providers are responsible for securing these applications. Customers can be normally responsible for operational security processes. However the following questions, along with other sections within this document, should assist in assessing their offerings:

• Administration controls are provided by them and can these be controls used to assign read and write privileges to other users?

• SaaS access control is quite fine grained and can be customized to ones organizations policy?

**2.2. Platform as a Service (PaaS)**

The consumer uses a hosting environment for their applications. The consumer controls the applications that run in the environment (and possibly has some control over the hosting environment), but does not control the operating system, hardware or network infrastructure on which they are running. The platform is typically an application framework. Generally speaking, PaaS service providers are responsible for the security of the platform software stack, and the recommendations throughout this document is a good foundation for ensuring a PaaS provider has considered security principles before designing and managing their PaaS platform. It is very difficult to get or obtain the detailed information from PaaS providers on exactly how they secure their platforms however there are some of the following questions that should be

along with other sections within these document.

• A high level description of containment and isolation measures is required for request information on how multi-tenanted applications are isolated from each other.

• What assurance can the PaaS provider give by which the data can be accessed?

• Does the provider ensure that the PaaS platform sandbox is monitored for new bugs, new attacks and vulnerabilities?

**2.3. Infrastructure as a Service (IaaS)**

The consumer uses “fundamental computing resources” such as processing power, storage, networking components or middleware. The consumer can control the operating system, storage, deployed applications and possibly networking components such as firewalls and load balancers, but not the cloud infrastructure beneath them., Many of the potential issues with personnel security arise because the IT infrastructure is under the control of a third party like traditional outsourcing, multiple customers get effect because of a physical security breach.

• What assurance can be provided to the customer regarding the physical security of the location?

• Who has unescorted access to IT infrastructure? For example, vendors’, managers, physical security staff, contractors, consultants, cleaners, etc.

• How often are access rights reviewed?

• How quickly can access rights be revoked?

• Does the security risks are assessed and parameters evaluated on a regular basis?

• How frequently?

• Are regular risk assessments being done which may include things such as neighboring buildings?

• Is access secure areas controled or monitored personnel (including third parties)?

• What are the policies/procedures that are used for loading, unloading and installing equipment?

• When are processes or procedures required to destroy old media or systems?

• Data overwritten?

• Physical destruction?

• How often are checks made to ensure compliance with the environment with the appropriate legal and regulatory requirements of a organization.

• **Public Cloud: A** public cloud is a standard cloud computing model wherein a service provider manages storage and computing resources on behalf of consumer over the Internet.

The term "public cloud" arose to differentiate between the standard model and the private cloud, which runs on proprietary network or data center of the user.

Public clouds are run by third parties, and applications from different users are shared on the provider’s cloud servers, storage systems, and networks. Public clouds are most often hosted away from customer premises, and they try to reduce customer risk and cost by substituting their enterprise infrastructure.

Applications which are required for temporary purpose or for short duration are the best suitable for deployment in a public cloud because it avoids the need to purchase additional equipment to solve a temporary need.

• **Private Cloud:** Private cloud (also called internal cloud or corporate cloud) is typically hosted on customer premises. With proprietary computing architecture, it provides hosted services to authorized users behind a company firewall. Thus company has control over resources, data, security and QoS.

The company owns the infrastructure and controls how applications are deployed on it.

Private clouds can be deployed in an organization datacenter or also at a collocation facility. Company’s own IT department or cloud service provider can built and manage private clouds. In this type of cloud computing, a company can install, configure, and operate the infrastructure as per its requirement and demand. A permanent application, or one that has specific requirements on quality of service or location of data, is most suitable to deploy in a private or hybrid cloud. Company’s own IT department uses their own private clouds for critical and other secured systems deployments

• **Hybrid cloud** is a cloud computing infrastructure composed of two or more clouds (private, community, or public) that remain unique entities but are bound together by standardized or proprietary technology that enables data and application portability

In hybrid cloud architecture, companies and individuals can obtain degrees of fault tolerance combined with locally immediate usability without internet connectivity. Y Hybrid Cloud architecture is the ideal combination that requires on premises resources and off site (remote) server based cloud infrastructure. Hybrid clouds do not have the flexibility, security and certainty of in-house applications.

**4. Security Issues in Private Cloud Computing**

Due primarily to the security concerns associated with the public cloud, many firms have elected to favor private cloud deployments over public clouds. While security pros are on their guard when it comes on private cloud. Private cloud gives more control to in house staff, but increased control cannot ignore the security. On the other hand, there are some security risks associated with all cloud models, private included. Because of security pros are less sensitive to risks and the control is high in the private model.

**4.1 Comingled regulatory environments**

Security cannot be fitted in every situation of IT environment. For example, that an entity regulated under PCI would find a non PCI certified environment is unacceptable for systems which are in cardholder data environment. This is true for both the public and private cloud.

An infrastructure is dedicated to be used alone does not mean everything can go with equal ease. Because private cloud grants greater control over regulatory compliance and security, the security should always be given the forefront of planning, particularly when multiple types of regulated data are in play, such as a customer data, comingled mix of payment card data and sensitive business intelligence.

**4.2 Viability of security tools**

When an organization virtualizes a physical host it always needs to evaluate how network aware tools will be impacted. If visibility into traffic can be impacted: network IDS and sniffers. For example, consider an n-tier Web application with separate Web, application and DB servers that attach to one switch that is monitored by IDS. If these three devices are moved

to virtual slices on a hypervisor, traffic will no longer visible on the wire, which will cause the IDS to lose visibility. At one time particularly true when large numbers of hosts are virtualized; more number of hosts’ means less time spent planning per host.

**4.3 Data expansion**

Cloud is a fantastic enabler of resource centralization. For example, a virtualized environment can allow far-flung resources to come together under an environment. However, if resources are centralized, data becomes denser. While this its a boon for management, it is challenging from security standpoint, particularly when considering tools are being used that operate across the data in aggregate. Antimalware scanning, bulk encryption and data discovery tools required that when we have a harder time dealing very large amounts of data. Existing tools should be examined to determine what impact they have on data volumes increase and new tools are considered when operation would be impacted severely and old tools are ineffective.

**4.4 Future proofing**

Private cloud does not means “on-premise,” but some may think that way. The defining aspect of private cloud is about which are users that use the infrastructure, not who maintains the infrastructure. So it is not necessarily many private cloud deployments will use on-premise infrastructure. And even if a deployment uses on-premise or dedicated resources today, that cannot prevent it from migrating off-premises to use a service provider or onto shared infrastructure. Organizations that put into a private environment today can easily migrate tomorrow. So private cloud deployments have many security advantages A private cloud deployment is every bit as serious as a move to public cloud and needs to be planned for accordingly.

**4.5 Fear of change.**

IT team may not be familiar with the term private clouds, so because of that there will be a big learning curve. There can also be new operational processes and some of old processes that need much of the rework. To turn this into a growth opportunity for people, the stress of doing and learning all this can be mitigated by helping your collogues keep in mind that these are important new skills in today's business environment.

**3.3 Proposed System Fundamentals concepts**

**3.4 Proposed Algorithms**

**4. SYSTEM ANALYSiS**

The **Systems Development Life Cycle (SDLC)**, or *Software Development Life Cycle* in [systems engineering](http://en.wikipedia.org/wiki/Systems_engineering" \o "Systems engineering), [information systems](http://en.wikipedia.org/wiki/Information_systems" \o "Information systems) and [software engineering](http://en.wikipedia.org/wiki/Software_engineering" \o "Software engineering), is the process of creating or altering systems, and the models and [methodologies](http://en.wikipedia.org/wiki/Methodologies" \o "Methodologies) that people use to develop these systems.

In software engineering the SDLC concept underpins many kinds of [software development methodologies](http://en.wikipedia.org/wiki/Software_development_methodologies" \o "Software development methodologies). These methodologies form the framework for planning and controlling the creation of an information system the [software development process](http://en.wikipedia.org/wiki/Software_development_process" \o "Software development process).

**SOFTWARE MODEL OR ARCHITECTURE ANALYSIS:**

Structured project management techniques (such as an SDLC) enhance management’s control over projects by dividing complex tasks into manageable sections. A software life cycle model is either a descriptive or prescriptive characterization of how software is or should be developed. But none of the SDLC models discuss the key issues like Change management, Incident management and Release management processes within the SDLC process, but, it is addressed in the overall project management. In the proposed hypothetical model, the concept of user-developer interaction in the conventional SDLC model has been converted into a three dimensional model which comprises of the user, owner and the developer. In the proposed hypothetical model, the concept of user-developer interaction in the conventional SDLC model has been converted into a three dimensional model which comprises of the user, owner and the developer. The ―one size fits all‖ approach to applying SDLC methodologies is no longer appropriate. We have made an attempt to address the above mentioned defects by using a new hypothetical model for SDLC described elsewhere. The drawback of addressing these management processes under the overall project management is missing of key technical issues pertaining to software development process that is, these issues are talked in the project management at the surface level but not at the ground level.

**WHAT IS SDLC?**

A software cycle deals with various parts and phases from planning to testing and deploying software. All these activities are carried out in different ways, as per the needs. Each way is known as a Software Development Lifecycle Model (SDLC). A software life cycle model is either a descriptive or prescriptive characterization of how software is or should be developed. A descriptive model describes the history of how a particular software system was developed. Descriptive models may be used as the basis for understanding and improving software development processes or for building empirically grounded prescriptive models.

**SDLC models** \* **The Linear model (Waterfall)** - Separate and distinct phases of specification and development. - All activities in linear fashion. - Next phase starts only when first one is complete. \* **Evolutionary development** - Specification and development are interleaved (Spiral, incremental, prototype based, Rapid Application development). - Incremental Model (Waterfall in iteration), **-** RAD(Rapid Application Development) **-** Focus is on developing quality product in less time, - **Spiral Model** - We start from smaller module and keeps on building it like a spiral. It is also called Component based development. \* **Formal systems development** - A mathematical system model is formally transformed to an implementation. \* **Agile Methods.** - Inducing flexibility into development. \* **Reuse-based development** - The system is assembled from existing components.

**The General Model**

Software life cycle models describe phases of the software cycle and the order in which those phases are executed. There are tons of models, and many companies adopt their own, but all have very similar patterns. Each phase produces deliverables required by the next phase in the life cycle. Requirements are translated into design. Code is produced during implementation that is driven by the design. Testing verifies the deliverable of the implementation phase against requirements.

**SDLC Methodology:**

**Spiral Model**

The spiral model is similar to the incremental model, with more emphases placed on risk analysis.  The spiral model has four phases: Planning, Risk Analysis, Engineering and Evaluation.  A\ software project repeatedly passes through these phases in iterations (called Spirals in this model).  The baseline spiral, starting in the planning phase, requirements is gathered and risk is assessed.  Each subsequent spirals builds on the baseline spiral. Requirements are gathered during the planning phase.  In the risk analysis phase, a process is undertaken to identify risk and alternate solutions.  A prototype is produced at the end of the  
risk analysis phase. Software is produced in the engineering phase, along with testing at  
the end of the phase.  The evaluation phase allows the customer to evaluate the output of the project to date before the project continues to the next spiral. In the spiral model, the angular component represents progress, and the radius of the spiral represents cost. Spiral Life Cycle Model.

This document play a vital role in the development of life cycle (SDLC) as it describes the complete requirement of the system. It means for use by developers and will be the basic during testing phase. Any changes made to the requirements in the future will have to go through formal change approval process.

SPIRAL MODEL was defined by Barry Boehm in his 1988 article, “A spiral Model of Software Development and Enhancement. This model was not the first model to discuss iterative development, but it was the first model to explain why the iteration models.

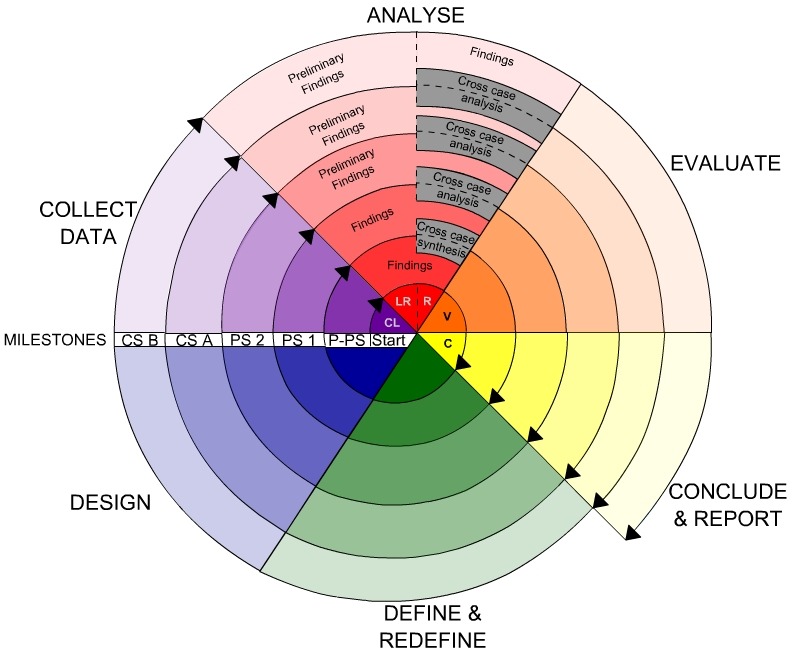
As originally envisioned, the iterations were typically 6 months to 2 years long. Each phase starts with a design goal and ends with a client reviewing the progress thus far. Analysis and engineering efforts are applied at each phase of the project, with an eye toward the end goal of the project.

The steps for Spiral Model can be generalized as follows:

* The new system requirements are defined in as much details as possible. This usually involves interviewing a number of users representing all the external or internal users and other aspects of the existing system.
* A preliminary design is created for the new system.
* A first prototype of the new system is constructed from the preliminary design. This is usually a scaled-down system, and represents an approximation of the characteristics of the final product.
* A second prototype is evolved by a fourfold procedure:

1. Evaluating the first prototype in terms of its strengths, weakness, and risks.
2. Defining the requirements of the second prototype.
3. Planning an designing the second prototype.
4. Constructing and testing the second prototype.

* At the customer option, the entire project can be aborted if the risk is deemed too great. Risk factors might involved development cost overruns, operating-cost miscalculation, or any other factor that could, in the customer’s judgment, result in a less-than-satisfactory final product.
* The existing prototype is evaluated in the same manner as was the previous prototype, and if necessary, another prototype is developed from it according to the fourfold procedure outlined above.
* The preceding steps are iterated until the customer is satisfied that the refined prototype represents the final product desired.
* The final system is constructed, based on the refined prototype.
* The final system is thoroughly evaluated and tested. Routine maintenance is carried on a continuing basis to prevent large scale failures and to minimize down time.



**Fig -Spiral Model**

**Advantages**

* High amount of risk analysis
* Good for large and mission-critical projects.
* Software is produced early in the software life cycle.

**4.1 Existing System:**

Several security schemes for data sharing on untrusted servers have been proposed. In these approaches, data owners store the encrypted data files in untrusted storage and distribute the corresponding decryption keys only to authorized users. Thus, unauthorized users as well as storage servers cannot learn the content of the data files because they have no knowledge of the decryption keys. However, the complexities of user participation and revocation in these schemes are linearly increasing with the number of data owners and the number of revoked users, respectively. By setting a group with a single attribute, Lu et al. proposed a secure provenance scheme based on the cipher text-policy attribute-based encryption technique, which allows any member in a group to share data with others. However, the issue of user revocation is not addressed in their scheme. Yu et al.presented a scalable and fine-grained data access control scheme in cloud computing based on the key policy attribute-based encryption (KP-ABE) technique. Unfortunately, the single owner manner hinders the adoption of their scheme into the case, where any user is granted to store and share data.

**Disadvantages:**

1. Without Guarantee of Identity Privacy, users may be unwilling to join in cloud computing systems.
2. Some users are behave as a misbehaving
3. It is not possible to trace which user converts as a false file.
4. Users are also modifying the data, that kind of possibility is available here.
5. The changes of membership make secure data sharing extremely difficult

**Proposed System:**

To solve the challenges presented above, we propose Mona, a secure multi-owner data sharing scheme for dynamic groups in the cloud. The main contributions of this paper include:

1. We propose a secure multi-owner data sharing scheme. It implies that any user in the group can securely share data with others by the untrusted cloud.

2. Our proposed scheme is able to support dynamic groups efficiently. Specifically, new granted users can directly decrypt data files uploaded before their participation without contacting with data owners. User revocation can be easily achieved through a novel revocation list without updating the secret keys of the remaining users. The size and computation

overhead of encryption are constant and independent with the number of revoked users.

3. We provide secure and privacy-preserving access control to users, which guarantees any member in a group to anonymously utilize the cloud resource. Moreover, the real identities of data owners can be revealed by the group manager when disputes occur.

4. We provide rigorous security analysis, and perform extensive simulations to demonstrate the efficiency of our scheme in terms of storage and computation overhead.

**Advantages:**

1. We provide the services as a low maintenance cost.
2. We provide the high quality services
3. Save the investments data
4. Minimize the complexity of secrete key management environment here.
5. Maintain the data as a sensitive data without accessing any unauthorized users.

**4.4 Modules Description:**

**Modules Description**

1. **System Model**
2. **Group Signature**
3. **Dynamic Broadcast Encryption**
4. **User Revocation**
5. **File Access and File Deletion**
6. **Traceability**

**System Model:**

Cloud is operated by CSPs and provides priced abundant storage services. However, the cloud is not fully trusted by users since the CSPs are very likely to be outside of the cloud users’ trusted domain. Similar to, we assume that the cloud server is honest but curious. That is, the cloud server will not maliciously delete or modify user data due to the protection of data auditing schemes, but will try to learn the content of the stored data and the identities of cloud users.

Group manager takes charge of system parameters generation, user registration, user revocation, and revealing the real identity of a dispute data owner. In the given example, the group manager is acted by the administrator of the company. Therefore, we assume that the group manager is fully trusted by the other parties.

Group members are a set of registered users that will store their private data into the cloud server and share them with others in the group. In our example, the staffs play the role of group members. Note that, the group membership is dynamically changed, due to the staff resignation and new employee participation in the company.

**Group Signature**

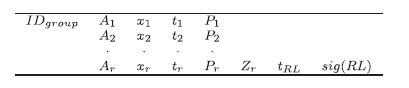
The concept of group signatures was first introduced in by Chaum and van Heyst. In general, a group signature scheme allows any member of the group to sign messages while keeping the identity secret from verifiers. Besides, the designated group manager can reveal the identity of the signature’s originator when a dispute occurs, which is denoted as traceability. In this paper, a variant of the short group signature scheme will be used to achieve anonymous access control, as it supports efficient membership revocation.

**Dynamic Broadcast Encryption**

Broadcast encryption enables a broadcaster to transmit encrypted data to a set of users so that only a privileged subset of users can decrypt the data. Besides the above characteristics, dynamic broadcast encryption also allows the group manager to dynamically include new members while preserving previously computed information, i.e., user decryption keys need not be recomputed, the morphology and size of cipher texts are unchanged and the group encryption key requires no modification. The first formal definition and construction of dynamic broadcast encryption are introduced based on the bilinear pairing technique in, which will be used as the basis for file sharing in dynamic groups.

**User Revocation**

User revocation is performed by the group manager via a public available revocation list (RL), based on which group members can encrypt their data files and ensure the confidentiality against the revoked users. As illustrated in Table 1, the revocation list is characterized by a series of time stamps (t1 < t2 < tr). Let ID group denote the group identity. The tuple (Ai; xi; ti) represents that user i with the partial private key (Ai; xi) is revoked at time ti. P1; P2; :::; Pr and Zr are calculated by the group manager with the private secret \_ as follows



**File Access and File Deletion**

Getting the data file and the revocation list from the cloud server. In this operation, the user first adopts its private key (A; x) to compute a signature sigma u on the message (IDgroup; IDdata; t) by using Algorithm 1, where t denote the current time, and the IDdata can be obtained from the local shared file list maintained by the manager. Then, the user sends a data request Containing (IDgroup; IDdata; t; sigma u) to the cloud server. Upon receiving the request, the cloud server employs Algorithm 2 to check the validity of the signature and performs revocation verification with Algorithm 3 if necessary according to the revocation list. After a successful verification, the cloud server responds the corresponding data file and the revocation list to the user. File stored in the cloud can be deleted by either the group manager or the data owner.

**Traceability**

When a data dispute occurs, the tracing operation is performed by the group manager to identify the real identity of the data owner.

**3.5 Feasibility Study**

Preliminary investigation examine project feasibility, the likelihood the system will be useful to the organization. The main objective of the feasibility study is to test the Technical, Operational and Economical feasibility for adding new modules and debugging old running system. All system is feasible if they are unlimited resources and infinite time. There are aspects in the feasibility study portion of the preliminary investigation:

* Technical Feasibility
* Operational Feasibility
* Economical Feasibility

**3.5.1 Economic Feasibility**

A system can be developed technically and that will be used if installed must still be a good investment for the organization. In the economical feasibility, the development cost in creating the system is evaluated against the ultimate benefit derived from the new systems. Financial benefits must equal or exceed the costs.

The system is economically feasible. It does not require any addition hardware or software. Since the interface for this system is developed using the existing resources and technologies available at NIC, There is nominal expenditure and economical feasibility for certain.

**3.5.2 Operational Feasibility**

Proposed projects are beneficial only if they can be turned out into information system. That will meet the organization’s operating requirements. Operational feasibility aspects of the project are to be taken as an important part of the project implementation. Some of the important issues raised are to test the operational feasibility of a project includes the following: -

* Is there sufficient support for the management from the users?
* Will the system be used and work properly if it is being developed and implemented?
* Will there be any resistance from the user that will undermine the possible application benefits?

This system is targeted to be in accordance with the above-mentioned issues. Beforehand, the management issues and user requirements have been taken into consideration. So there is no question of resistance from the users that can undermine the possible application benefits.

The well-planned design would ensure the optimal utilization of the computer resources and would help in the improvement of performance status.

**3.5.3 Technical Feasibility**

The technical issue usually raised during the feasibility stage of the investigation includes the following:

* Does the necessary technology exist to do what is suggested?
* Do the proposed equipments have the technical capacity to hold the data required to use the new system?
* Will the proposed system provide adequate response to inquiries, regardless of the number or location of users?
* Can the system be upgraded if developed?
* Are there technical guarantees of accuracy, reliability, ease of access and data security?

Earlier no system existed to cater to the needs of ‘Secure Infrastructure Implementation System’. The current system developed is technically feasible. It is a web based user interface for audit workflow at NIC-CSD. Thus it provides an easy access to the users. The database’s purpose is to create, establish and maintain a workflow among various entities in order to facilitate all concerned users in their various capacities or roles. Permission to the users would be granted based on the roles specified. Therefore, it provides the technical guarantee of accuracy, reliability and security. The software and hard requirements for the development of this project are not many and are already available in-house at NIC or are available as free as open source. The work for the project is done with the current equipment and existing software technology. Necessary bandwidth exists for providing a fast feedback to the users irrespective of the number of users using the system.

* 1. **System Requirements Specification**

**5.1 Introduction**

A **Software Requirements Specification** (**SRS**) – a [requirements specification](http://en.wikipedia.org/wiki/Requirements_specification" \o "Requirements specification) for a [software system](http://en.wikipedia.org/wiki/Software_system" \o "Software system) – is a complete description of the behaviour of a system to be developed. It includes a set of [use cases](http://en.wikipedia.org/wiki/Use_case" \o "Use case) that describe all the interactions the users will have with the software. In addition to use cases, the SRS also contains non-functional requirements. [Non-functional requirements](http://en.wikipedia.org/wiki/Non-functional_requirements" \o "Non-functional requirements) are requirements which impose constraints on the design or implementation (such as [performance engineering](http://en.wikipedia.org/wiki/Performance_engineering" \o "Performance engineering) requirements, [quality](http://en.wikipedia.org/wiki/Quality_(business)" \o "Quality (business)) standards, or design constraints).

**System requirements specification:** A structured collection of information that embodies the requirements of a system. A [business analyst](http://en.wikipedia.org/wiki/Business_analyst" \o "Business analyst), sometimes titled [system analyst](http://en.wikipedia.org/wiki/System_analyst" \o "System analyst), is responsible for analyzing the business needs of their clients and stakeholders to help identify business problems and propose solutions. Within the [systems development life cycle](http://en.wikipedia.org/wiki/Systems_development_life_cycle" \o "Systems development life cycle) domain, typically performs a liaison function between the business side of an enterprise and the information technology department or external service providers. Projects are subject to three sorts of requirements:

* **[Business requirements](http://en.wikipedia.org/wiki/Business_requirements" \o "Business requirements)** describe in business terms *what* must be delivered or accomplished to provide value.
* **Product requirements** describe properties of a system or product (which could be one of

several ways to accomplish a set of business requirements.)

* **Process requirements** describe activities performed by the developing organization. For instance, process requirements could specify specific methodologies that must be followed, and constraints that the organization must obey.

Product and process requirements are closely linked. Process requirements often specify the activities that will be performed to satisfy a product requirement. For example, a maximum development cost requirement (a process requirement) may be imposed to help achieve a maximum sales price requirement (a product requirement); a requirement that the product be maintainable (a Product requirement) often is addressed by imposing requirements to follow particular development styles

**5.2 Purpose**

An systems engineering, a **requirement** can be a description of *what* a system must do, referred to as a [Functional Requirement](http://en.wikipedia.org/wiki/Functional_requirements" \o "Functional requirements). This type of requirement specifies something that the delivered system must be able to do. Another type of requirement specifies something about the system itself, and how well it performs its functions. Such requirements are often called [Non-functional requirements](http://en.wikipedia.org/wiki/Non-functional_requirements" \o "Non-functional requirements), or 'performance requirements' or 'quality of service requirements.' Examples of such requirements include usability, availability, reliability, supportability, testability and maintainability.

A collection of requirements define the characteristics or features of the desired system. A 'good' list of requirements as far as possible avoids saying *how* the system should implement the requirements, leaving such decisions to the system designer. Specifying how the system should be implemented is called "implementation bias" or "solution engineering". However, *implementation constraints* on the solution may validly be expressed by the future owner, for example for required interfaces to external systems; for interoperability with other systems; and for commonality (e.g. of user interfaces) with other owned products.

In software engineering, the same meanings of requirements apply, except that the focus of interest is the software itself.

**5.3 Functional Requirements:**

**Modules:**

1. Owner and User
2. Manager
3. Registration

**Functional Requirements:**

**Owner and user :**

Upload file

View files

Generate key

Upload File

View All Uploaded Files

View groups

Sent Change Request to user groups

View group change request.

View Groups signature

Download File

Generate Key signature Graph

Generate Key Storage Graph

Time and Cost Graph

**Manager:**

* View Groups. groups user, user uploaded files
* View Request Change Users
* View All User Details
* View Uploaded Files with signatures and keys.

Registration:

* Home Page
* User. Cloud server owner and Owner Login
* User, Owner registration.

**5.4 Non Functional Requirements**

The major non-functional Requirements of the system are as follows

**Usability**

The system is designed with completely automated process hence there is no or less user intervention.

**Reliability**

The system is more reliable because of the qualities that are inherited from the chosen platform java. The code built by using java is more reliable.

**Performance**

This system is developing in the high level languages and using the advanced front-end and back-end technologies it will give response to the end user on client system with in very less time.

**Supportability**

The system is designed to be the cross platform supportable. The system is supported on a wide range of hardware and any software platform, which is having JVM, built into the system.

**5.5 Software Requirements:**

**Software Requirements Specification:**

**Software Requirements:**

Language : JDK (1.7.0)

Frontend : Swings

Backend : oracle Database

IDE : my eclipse 8.6

Operating System : windows XP

**Hardware Requirements:**

Processor : Pentium IV

Hard Disk : 80GB

RAM : 2GB

1. **System Design**

**6.1 Introduction**

The purpose of the design phase is to plan a solution of the problem specified by the requirement document. This phase is the first step in moving from the problem domain to the solution domain. In other words, starting with what is needed, design takes us toward how to satisfy the needs. The design of a system is perhaps the most critical factor affection the quality of the software; it has a major impact on the later phase, particularly testing, maintenance. The output of this phase is the design document. This document is similar to a blueprint for the solution and is used later during implementation, testing and maintenance. The design activity is often divided into two separate phases System Design and Detailed Design.

System Design also called top-level design aims to identify the modules that should be in the system, the specifications of these modules, and how they interact with each other to produce the desired results. At the end of the system design all the major data structures, file formats, output formats, and the major modules in the system and their specifications are decided.

During, Detailed Design, the internal logic of each of the modules specified in system design is decided. During this phase, the details of the data of a module is usually specified in a high-level design description language, which is independent of the target language in which the software will eventually be implemented.

In system design the focus is on identifying the modules, where as during detailed design the focus is on designing the logic for each of the modules. In other works, in system design the attention is on what components are needed, while in detailed design how the components can be implemented in software is the issue.

Design is concerned with identifying software components specifying relationships among components. Specifying software structure and providing blue print for the document phase. Modularity is one of the desirable properties of large systems. It implies that the system is divided into several parts. In such a manner , the interaction between parts is minimal clearly specified.

During the system design activities , Developers bridge the gap between the requirements specification , produced during requirements elicitation and analysis , and the system that is delivered to the user.

Design is the place where the quality is fostered in development . Software design is a process through which requirements are translated into a representation of software.

**6.2 System Model**

**Introduction to UML**

The unified Modeling Language (UML) is a standard language for writing software blueprints. The UML may be used to visualize, specify , construct and document the artifacts of software-intensive system.

The goal of UML is to provide a standard notation that can be used by all object - oriented methods and to select and integrate the best elements .UML is itself does not prescribe or advice on how to use that notation in a software development process or as part of an object - design methodology. The UML is more than just bunch of graphical symbols. Rather , behind each symbol in the UML notation is well-defined semantics.

The system development focuses on three different models of the system.

* Functional model
* Object model
* Dynamic model

**Functional model**  in UML is represented with use case diagrams , describing the functionality of the system from user point of view.

**Object model** in UML is represented with class diagrams , describing the structure of the system in terms of objects , attributes , associations and operations.

**Dynamic model**  in UML is represented with sequence diagrams , start chart diagrams and activity diagrams describing the internal behaviour of the system.

**6.3 Scenarios**

A Use Case is an abstraction that all describes all possible scenarios involving the described functionality. A scenario is an instance of a use case describing a concrete set of actions.

* The **name** of the scenario enables us to refer it ambiguously. The name of scenario is underlined to indicate it is an instance.
* The **Participating actor instance** field indicates which actor instance are involved in this scenario. Actor instance also have underlined names.
* The **Flow of Events** of scenario describe the sequence of events step by step.

**6.3.1 Use Case Model**

Use case diagrams represent the functionality of the system from a user point of view. A Use case describes a function provided by the system that yields a visible result for an actor. an actor describe any entity that interacts with the system. The identification of actors and use cases results in the definition of the boundary of the system, which is , in differentiating the tasks accomplished by the system and the tasks accomplished by its environment. The actors outside the boundary of the system, where as the use cases are inside the boundary of the system

A Use case contains all the events that can occur between an actor and a set of scenarios that explains the interactions as sequence of happenings.

**Actors**

Actors represent external entities that interact with the system. An actor can be human or external system.

Actor are not part of the system. They represent anyone or anything that interact with the system.

An Actor may

* Only input information to the system.
* Only receive information from the system.
* Input and receive information from to and from the system.

During this activity , developers indentify the actors involved in this system are:

**User:**

User is an actor who uses the system and who performs the operations like data classifications and execution performance that are required for him.

**Use Cases:**

Use cases are used during requirements elicitation and analysis to represent the functionality of the system. Use case focus on the behaviour of the system from an external point of view. The identification of actors and use cases results in the definition of the boundary of the system , which is , in differentiating the tasks accomplished by the system and the tasks accomplished by its environment. The actors are outside the boundary of the system , where as the use cases are inside the boundary of the system.

Use case Diagram



**6.3.2 Object model**

**Class Diagram**

Class Diagrams are used to describe the structure of the system. Classes are abstractions that specify the common structure and behaviour of a set of objects. Objects are instances of classes that are created , modified and destroyed during the execution of a system. An object has state that includes the values of its attributes and links with other objects.

The class diagram is used to refine the use cases diagrams and define a detailed design of the system. The class diagram classifies the actors defined in the use case diagram into a set of interrelated classes. The relationship or association between the classes can be either an "is-a" or "has-a" relationship. Each class in the class diagram may be capable of providing certain functionalities. These functionalities provided by the class are termed "methods" of the classes. Apart from this , each class may have certain "attributes" that uniquely indentify the class. In the class diagram these classes are represented with boxes which contain three parts.

* The upper part holds the name of the class
* The middle part contains the attribute of the class.
* The bottom part gives the methods or operations the class can take or undertake.



Class Diagram

**6.3.3 Dynamic model**

**6.3.3.1 Sequence Diagram**

Sequence diagrams are used to formalize the dynamic behaviour of the system and to visualize the communication among the objects. They are useful for identifying the additional objects that participate in the use case. Sequence diagram represent the objects participating in the interaction horizontally and time vertically.

Sequence diagrams typically show a user or actor and the objects and the components they interact with the execution of the use case. Each column represent an objects that participate in the interaction. Message is shown by solid arrows. Labels on the solid arrows represent the message names. Activations are depicted by vertical rectangles. The actor who initiates the interaction is shown in the left most columns . The messages coming from the actor represent the interactions described in the use case diagrams.



Sequence Diagram

Collaboration Diagram:



**6.3.3.2 State Chart Diagram**

UML State chart is notation for describing the sequence of states an object goes through in response to external events. Objects have behaviour and state. The state of an object depends on its current activity or condition. A state chart diagram shows the possible states of the object ad the transitions that cause a change in state.

State chart describes the dynamic behaviour of an individual object as a number of states. A state is a condition satisfied by attributes of objects. Given a state , a transition represents a future state the object can move to and the conditions associated with the change of state.

A state is depicted by a rounded rectangle A transition is depicted by open arrows connecting two states. States are labeled with their names. A small solid black circle indicates the initial state and a circle surrounding the small solid circle indicates the final state.

State Chart Diagram:



View User Groups

Change group User

View All Cloud Files

View All Users

View Group Change Request

State Chart Diagram

**6.3.3.3 Activity Diagram**

An Activity diagram describes the behaviour of the system in terms of activities. Activities are modeling elements that represent the execution of set of operations. The completion of these operations triggers a transition to another activity. Activity diagrams similar to flowchart diagrams in that they can be used to represent control flow and data flow . Activities are represented by rounded rectangles and arrows are represented transition between activities . Think bars represent the synchronization of the control flow.

Activity Diagram:



Activity Diagram

Component Diagram:



**Deployment Diagram:**



**6.4 Data Flow Diagrams:**

A graphical tool used to describe and analyze the moment of data through a system manual or automated including the process, stores of data, and delays in the system. Data Flow Diagrams are the central tool and the basis from which other components are developed. The transformation of data from input to output, through processes, may be described logically and independently of the physical components associated with the system. The DFD is also know as a data flow graph or a bubble chart.

DFDs are the model of the proposed system. They clearly should show the requirements on which the new system should be built. Later during design activity this is taken as the basis for drawing the system’s structure charts. The Basic Notation used to create a DFD’s are as follows:

**1. Dataflow:** Data move in a specific direction from an origin to a destination.

**2. Process:** People, procedures, or devices that use or produce (Transform) Data. The physical component is not identified.

**3. Source:** External sources or destination of data, which may be People, programs, organizations or other entities.

**4. Data Store:** Here data are stored or referenced by a process in the System.

5.rombous: decision

**CONTEXT LEVEL DIAGRAM**

**Context Level0 DFD**



**Context level1 Diagram:**

**Login DFD**



**Context level 2:**

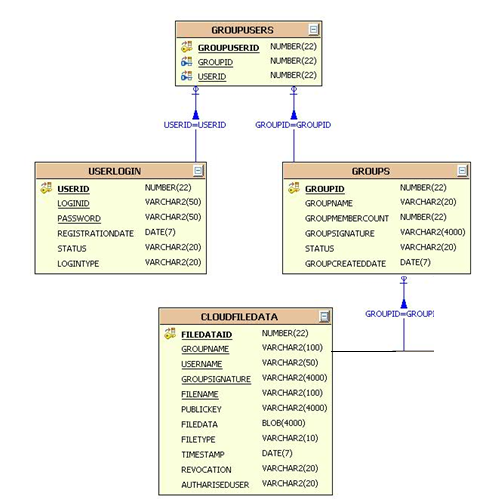
s



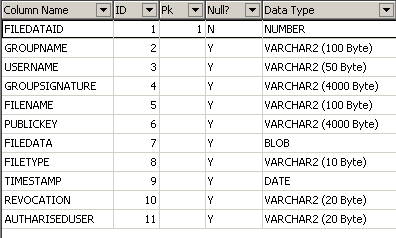
**ER Diagram:**

In [software engineering](http://en.wikipedia.org/wiki/Software_engineering" \o "Software engineering), an entity-relationship model (ERM) is an abstract and conceptual representation of [data](http://en.wikipedia.org/wiki/Data" \o "Data). Entity-relationship modeling is a [database modeling](http://en.wikipedia.org/wiki/Database_model" \o "Database model) method, used to produce a type of [conceptual schema](http://en.wikipedia.org/wiki/Conceptual_schema" \o "Conceptual schema) or [semantic data model](http://en.wikipedia.org/wiki/Semantic_data_model" \o "Semantic data model) of a system, often a [relational database](http://en.wikipedia.org/wiki/Relational_database" \o "Relational database), and its requirements in a [top-down](http://en.wikipedia.org/wiki/Top-down" \o "Top-down) fashion. Diagrams created by this process are called entity-relationship diagrams, ER diagrams, or ERDs. The definitive reference for entity-relationship modeling is [Peter Chen](http://en.wikipedia.org/wiki/Peter_Chen" \o "Peter Chen)'s 1976 paper. However, variants of the idea existed previously, and have been devised subsequently. An [entity](http://en.wikipedia.org/wiki/Entity" \o "Entity) may be defined as a thing which is recognized as being capable of an independent existence and which can be uniquely identified. An entity is an abstraction from the complexities of some domain. When we speak of an entity we normally speak of some aspect of the real world which can be distinguished from other aspects of the real world. An entity may be a physical object such as a house or a car, an event such as a house sale or a car service, or a concept such as a customer transaction or order. Although the term entity is the one most commonly used, following Chen we should really distinguish between an entity and an entity-type. An entity-type is a category. An entity, strictly speaking, is an instance of a given entity-type. There are usually many instances of an entity-type. Because the term entity-type is somewhat cumbersome, most people tend to use the term entity as a synonym for this term. Entities can be thought of as [nouns](http://en.wikipedia.org/wiki/Noun" \o "Noun). Examples: a computer, an employee, a song, a mathematical theorem. A relationship captures how two or more entities are related to one another. Relationships can be thought of as [verbs](http://en.wikipedia.org/wiki/Verb" \o "Verb), linking two or more nouns. Examples: an *owns* relationship between a company and a computer, a *supervises* relationship between an employee and a department, a *performs* relationship between an artist and a song, a *proved* relationship between a mathematician and a theorem. The model's linguistic aspect described above is utilized in the [declarative](http://en.wikipedia.org/wiki/Declarative_programming_language" \o "Declarative programming language) database [query language](http://en.wikipedia.org/wiki/Query_language" \o "Query language) [ERROL](http://en.wikipedia.org/wiki/ERROL" \o "ERROL), which mimics [natural language](http://en.wikipedia.org/wiki/Natural_language" \o "Natural language) constructs. Entities and relationships can both have attributes. Examples: an *employee* entity might have a *Social Security Number* (SSN) attribute; the *proved* relationship may have a *date* attribute. Every entity (unless it is a [weak entity](http://en.wikipedia.org/wiki/Weak_entity" \o "Weak entity)) must have a minimal set of uniquely identifying attributes, which is called the entity's [primary key](http://en.wikipedia.org/wiki/Primary_key" \o "Primary key). Entity-relationship diagrams don't show single entities or single instances of relations. Rather, they show entity sets and relationship sets. Example: a particular *song* is an entity. The collection of all songs in a database is an entity set. The *eaten* relationship between a child and her lunch is a single relationship. The set of all such child-lunch relationships in a database is a relationship set. In other words, a relationship set corresponds to a [relation in mathematics](http://en.wikipedia.org/wiki/Relation_(mathematics)" \o "Relation (mathematics)), while a relationship corresponds to a member of the relation. Certain [cardinality constraints](http://en.wikipedia.org/wiki/Cardinality_(data_modeling)" \o "Cardinality (data modeling)) on relationship sets may be indicated as well.

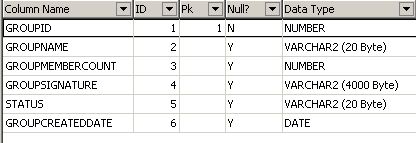
ER Diagram



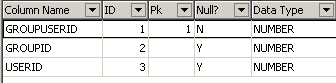
CLOUDFILEDATA



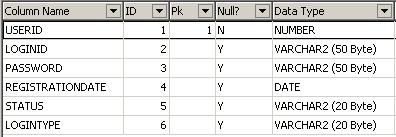
GROUPS



GROUPUSERS



USERLOGIN



1. **Implementation**

**7.1 Introduction**

Implementation is the stage where the theoretical design is turned in to working system. The most crucial stage is achieving a new successful system and in giving confidence on the new system for the users that it will work efficiently and effectively.

The system can be implemented only after through testing is done and if it found to work according to the specification. It involves careful planning, investigation of the current system and its constraints on implementation, design of methods to achieve the change over and an evaluation of change over methods a part from planning. Two major tasks of preparing the implementation are education and training of the users and testing of the system.

The implementation phase comprises of several activities. The required hardware and software acquisition is carried out. The System may require some hardware and software acquisition is carried out. The system may require some software to be developed. For this, programs are written and tested. The user then changes over to his new fully tested system and the old system is discontinued.

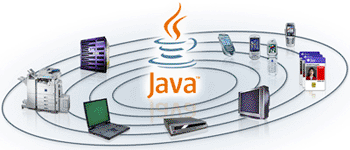
Implementation is the process of having systems personnel check out and put new equipment in to use, train users, install the new application, and construct any files of data needed to it.

Depending on the size of the organization that will be involved in using the application and the risk associated with its use, system developers may choose to test the operation in only one area of the firm, say in one department or with only one or two persons. Sometimes they will run the old and new systems together to compare the results. In still other situations, developers will stop using the old system one-day and begin using the new one the next. As we will see, each implementation strategy has its merits, depending on the business situation in which it is considered. Regardless of the implementation strategy used, developers strive to ensure that the system’s initial use in trouble-free.

**7.1 Technology Description**

**About the Java Technology**

The Java platform consists of the Java application programming interfaces (APIs) and the Java virtual machine (JVM).



The following Java technology lets developers, designers, and business partners develop and deliver a consistent user experience, with one environment for applications on mobile and embedded devices. Java meshes the power of a rich stack with the ability to deliver customized experiences across such devices.

Java APIs are libraries of compiled code that you can use in your programs. They let you add ready-made and customizable functionality to save you programming time.  
Java programs are run (or interpreted) by another program called the Java Virtual Machine. Rather than running directly on the native operating system, the program is interpreted by the Java VM for the native operating system. This means that any computer system with the Java VM installed can run Java programs regardless of the computer system on which the applications were originally developed.

In the Java programming language, all source code is first written in plain text files ending with the .java extension. Those source files are then compiled into .class files by the javac compiler. A .class file does not contain code that is native to your processor; it instead contains bytecodes — the machine language of the Java Virtual Machine (Java VM). The java launcher tool then runs your application with an instance of the Java Virtual Machine.

Because the Java VM is available on many different operating systems, the same .class files are capable of running on Microsoft Windows, the Solaris TM Operating System (Solaris OS), Linux, or Mac OS.

Java technology is both a programming language and a platform.

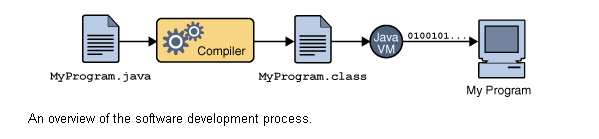
**The Java Programming Language**

The Java programming language is a high-level language that can be characterized by all of the following buzzwords:

|  |  |
| --- | --- |
| * Simple * Object oriented * Distributed * Multithreaded * Dynamic | * Architecture neutral * Portable * High performance * Robust * Secure |

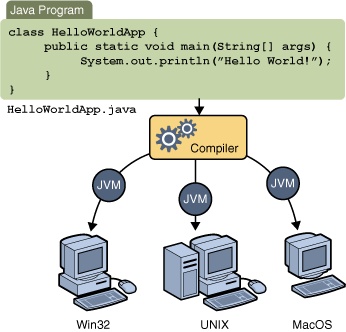
Each of the preceding buzzwords is explained in *[The Java Language Environment](http://java.sun.com/docs/white/langenv/" \t "_blank)* , a white paper written by James Gosling and Henry McGilton.

In the Java programming language, all source code is first written in plain text files ending with the .java extension. Those source files are then compiled into .class files by the javac compiler. A .class file does not contain code that is native to your processor; it instead contains *bytecodes* — the machine language of the Java Virtual Machine[1](http://download.oracle.com/javase/tutorial/getStarted/intro/definition.html" \l "FOOT) (Java VM). The java launcher tool then runs your application with an instance of the Java Virtual Machine.



An overview of the software development process.

Because the Java VM is available on many different operating systems, the same .class files are capable of running on Microsoft Windows, the Solaris™ Operating System (Solaris OS), Linux, or Mac OS. Some virtual machines, such as the [Java HotSpot virtual machine](http://java.sun.com/products/hotspot/" \t "_blank), perform additional steps at runtime to give your application a performance boost. This include various tasks such as finding performance bottlenecks and recompiling (to native code) frequently used sections of code



Through the Java VM, the same application is capable of running on multiple platforms.

**AWT**

AWT is the acronym of Abstract Window Toolkit.

**Introduction:**

GUI (Graphical User Interface) is a mediator (translator) between the end user and the program.

**Form :** It is a collection of GUIs designed for one specific purpose.

\* Console is a program, provided by the local OS, which supports only text output.

\* In order to display GUI outputs, we have another program (similar to the console) provided by the local os. This program is, window.

\* All the java classes, which are used to create GUIs are present in the package java.awt.

\* The theory behind the creation of GUIs is as follows.

\* The functionality of the constructor of the class, whose GUI is to be created, is to create that particular GUI.

\* Strictly speaking, this is not the functionality of the constructor of that class.

\* In fact, the constructor has some statements which are nothing but, a call to a specific file in the local os. This particular file will be responsible in creating the corresponding GUI. In the foreground, it seems as if the GUI is created just because of the constructor.

\* Whatever language we use to create the GUIs, (for example : java, html, vb e.t.c.) in all these cases, only the files present in the local os will be responsible for creating the corresponding GUIs.

\* Because of this, the GUIs created in all these cases will look alike.

\* Let us take java in particular.

\* When we want to create a GUI, we need to create object of that class which represents the corresponding GUI. Let us assume we need to create a GUI – a button. Then we need to write the following code, i.e.

\* Button b1 = new Button ();

\* So, to create the button GUI, JVM executes the constructor of the Button class. Inside the constructor we have statements which execute the files present in the local os to create the GUI.

\* The means, we are making JVM to execute the files present in the local operating system.

\* But remember that, JVM can execute only byte code, which is generated from the .class file and this .class file is results from a .java file.

\* The files present in the local operating system need not be in java and definitely they will not be in java.

\* They will be in some other native language.

**Question :** Then, how can JVM execute the files present in the local os to create the GUIs?

**Ans:** JVM depends upon some classes known as Peer Classes to execute the files present in the local os.

\* Thus, every awt component depends upon the files of the local os to create GUIs, making JVM to execute those files which are non-java files. This increases the overhead on the JVM. This is the reason why we call awt classes (components) as heavy-weight components

\* From the above discussion, it is clear that each os will have its own files for creating the GUIs.

\* Now, if we create a GUI using java on one os, then the look and feel of this GUI will be completely different when we execute the same java program on another OS.

**Question: Java is not purely platform independent. Justify.**

**Ans:** Because of the dependency of awt programs on the local os in creating GUIs, the outputs of these programs very from one os to another os. i.e. the outputs of the awt programs are platform dependent. This makes java to be platform-dependent.

This is the reason, why we say that java is not purely platform-independent.

**Disadvantages of AWT :**

(i) The AWT programs increase the overhead on the JVM.

(ii) The make java to be platform dependent.

\* The above disadvantages are over-come in Swings

\* Swings is an extension of AWT. The swing components do not depend upon the files of the local os in creating the GUIs and thereby, they decrease the overhead on the JVM.

\* Thus, swings are known as Light-Weight Components.

\* The following is the class hierarchy of java.awt package.



NOTE: Applet class does not belong to java.awt package.

\* Applet class belongs to java. applet package.

\* We know that window is used for displaying the GUIs.

\* A blind rule is that, when we want to add a component to another component then the later component should have the properties of a container.

\* Whenever we want to view a GUI output, we add all the required components to the window and then they are displayed that means, window has the properties of a container.

\* From the class hierarchy, it is clear that window is the sub class of container class. Hence it has all the properties of a container.

\* If we want to add a Button to another button, it is not possible because, button is not a sub class of container and it does not have the properties of a container.

\* Same is the case with the other classes present in that level of hierarchy.

\* Similarly, we can not think of adding Text Filed to a Label, Text Area to a List e.t.c.

\* Thus, any component which has the properties of container is eligible to accommodate other components in it.

\* It is clear from the class hierarchy that panel is a sub-class of container and window is at the same level of hierarchy.

\* Panel has the properties of container and any no. of components can be added to the panel. To make them visible on the screen, we again need window, but observe that panel does not have the properties of a window.

\* Therefore, panel should be added to the window to make the components visible.

\* Similarly, we can create a Frame. Frame has the properties of window as well as container

\* As Button, Text Area e.t.c. have some properties, window also has some properties.



- It has a title-bar

- It has some system Menus

- We should be able to drag it

- We should be able to resize it e.t.c.

\* Some of the best examples of the window application are Edit plus Editor, Notepad, IE browser e.t.c.

**NOTE:** As console cannot display GUIs, window cannot display text directly.

\* In order to display text on the window, we need to represent the text in the form of a GUI component and then add it to the window. For this, we make use of the Label class.

\* We always prefer to create a window through Frame because window alone has no feature of boundaries. The facility of having boundaries for a window is possible with Frame. This is the reason why we create window through Frame.

\* Using the functions of the Frame class we can set the dimensions (size) for the window.

**Ex:** f.setsize (400, 400);

**NOTE:** By default, window would always be created in the invisible mode. To make the window visible on the VDU, we should explicitly set the visibility property of the window to the Boolean value ‘true’.

**Ex:** f = set visible (true);

\* A constructor defined in the Frame class takes a string class object as an argument. Thus, whatever the text we want to see in the title bar of the window, we need to pass it as an argument (in the form of a string class object) to the constructor of the Frame class.

Let us develop a simple program which displays a GUI as shown aside.



**Hint:** This GUI consists of six components

 two labels + two text fields + one button + one frame (window)

import java.awt.\*;

class Form1 Form1.java

{

public static void main (string args [])

{

Frame f1 = new Frame (“Login Page”);

Label l1 = new Label (“User Name”);

Label l2 = new Label (“Password”);

Text Field tf1 = new Text Field ();

Text Field tf2 = new Text Field ();

Button b1 = new Button (“submit”);

f1. add (l1);

f1. add (l2);

f1. add (tf1);

f1. add (tf2);

f1. add (b1);

f1. set size (437, 437);

f1. set visible (true);

}

}



\* When we execute the above program, we can see only the button on the window. This is because, all the components have been added, but one upon another. The last component added is the button and hence we see only the button.

\* This resulted because we have not mentioned, to which part of the window, the components have to be added. Here they have occupied the entire window.

\* So, to make all the components visible we have to align them properly on the window.

\* We have some standard procedures using which we can align the components in the frame (container)

\* The standard procedures which exactly specify the way in which we align the components on the container are known as Layout Managers.

\* We have five Layout Managers in the awt package, where each Layout Manager is the name of a class.

(i) Border Layout (ii) Flow Layout

(iii) Grid Layout (iv) Grid Bag Layout

(v) Card Layout

\* Therefore, we should mention the layout according to which the components are to be aligned on the container.

\* We can mention the layout (specification or procedure) byusing the set Layout () method of the container class. This method is a non-static method.

\* The set Layout () method takes the object of any of the above five layouts as an argument, according to which the components would be aligned (arranged) on the containers.

**Border Layout:**



\* According to the specifications of the Border Layout, the container area will be divided into five parts as shown in the fig.

\* The Border Layout class has some static constants to refer to these areas as Border Layout. SOUTH e.t.c.

\* The default layout associated with the Frame class object is Border Layout. The default location associated with the Border Layout is CENTER.

\* Therefore, when we call add () method on the Frame class object (i.e. f. add (xxx)), it takes the layout to be Border Layout and adds the component to the centre of it.

\* An important property of the Border Layout is, when a component is added to the CENTER, then the component not only occupies the CENTER but also stretches to the other four areas, if there are not components in those areas.

\* For example, if we add a button to the center, then it occupies the entire frame.

\* If we add two buttons – one to the NORTH and one to the center, then the button which is added to the CENTER also stretches to the EAST, WEST and SOUTH areas.

**Flow Layout:**

\* According to this layout, the container will be divided into rows and columns depending upon the components and their size.

\* When we start adding components to the frame according to Flow Layout, then, the components would be added row-wise i.e. first row  column-1, column-2, e.t.c. If the row is filled, then the next component would be added to first column of second row and so on.

**NOTE:** When we specify the layout as Flow Layout, then, initially it assumes the entire space as 1 row & 1 column.

\* When we add a component it would be added in the centre at the top (because initially it is 1 row & 1 column).

\* Now, when we add another component, control check whether this component can be accommodated to the right of the existing component.

\* It yes, it will be added, thus creating a second column.

\* If no, the component would be added in the next row.

\* Hence, size of the column completely depends upon the size of the component being added.

\* Therefore, it is obvious that, when we add components to the container (frame) according to Flow Layout, then the columns will be dynamically generated.

**NOTE:** The no. of rows and columns may or may not be equal. i.e. the no. of columns may vary from one row to another row, depending upon the size of the components being added.

**NOTE:** When we resize the frame (on which the components are arranged according to flow layout) then the columns are going to get adjusted accordingly and even the components are going to get adjusted accordingly.

**7.2 Code Snippets (Logics) & Analysis**

File Upload and Downlaod :

String status = registrationDAO.getFileStatus(jList1.getSelectedValue()

.toString());

**if** (status.equals("false")) {

JOptionPane.*showMessageDialog*(jPanel,

"wait for Cloud Server provider permission");

} **else** {

String privatekey = JOptionPane

.*showInputDialog*("Please Enter Private Key");

**if** (privatekey != **null**) {

String existedkey = registrationDAO.getKey(jList1

.getSelectedValue().toString());

**if** (existedkey.trim().equals(privatekey.trim())) {

JOptionPane.*showMessageDialog*(**null**, "Key Matched",

"Private Key", 1);

Users users = (Users) treeMap.get(count);

**if** (!(users.getDownloadtime() > users.getMintime())) {

**try** {

String downloadfile = registrationDAO

.downloadFile(jList1.getSelectedValue()

.toString());

**if** (!downloadfile.equals("")) {

JFileChooser c = **new** JFileChooser();

**int** x = c.showSaveDialog(**null**);

**if** (x == JFileChooser.*APPROVE\_OPTION*) {

**try** {

String saveFilename = **null**;

String filename = **new** File(downloadfile)

.getName().toString();

System.*out*.println(filename);

String[] token = filename.split("\\.");

**if** (c.getSelectedFile() != **null**) {

**try** {

saveFilename = c

.getSelectedFile()

+ "." + token[1];

FileInputStream fileInputStream = **new** FileInputStream(

**new** File(downloadfile));

FileOutputStream fileOutputStream = **new** FileOutputStream(

saveFilename);

**int** data;

**while** ((data = fileInputStream

.read()) != -1) {

fileOutputStream

.write(data);

// fileOutputStream.write((int)

// data);

}

} **catch** (Exception e) {

// **TODO**: handle exception

}

} **else** {

saveFilename = **new** File(

downloadfile).getName()

.toString();

}

System.*out*.println(saveFilename);

} **catch** (Exception e) {

e.printStackTrace();

}

}

}

} **catch** (Exception e) {

System.*out*.println(e);

}

} **else** {

JOptionPane.*showMessageDialog*(**null**,

"Time is not sufficient left time to download :"

+ users.getDifftime() + "secs",

"Private Key", 1);

}

}

**else**

JOptionPane.*showMessageDialog*(**null**, "Key Not Matched",

"Private Key", 1);

}

**8. System Testing**

**8.1 Testing Methodologies**

Testing is the process of finding differences between the expected behavior specified by system models and the observed behavior implemented system. From modeling point of view , testing is the attempt of falsification of the system with respect to the system models. The goal of testing is to design tests that exercise defects in the system and to reveal problems.

The process of executing a program with intent of finding errors is called testing. During testing , the program to be tested is executed with a set of test cases , and the output of the program for the test cases is evaluated to determine if the program is performing as expected . Testing forms the first step in determining the errors in the program. The success of testing in revealing errors in program depends critically on test cases.

**Strategic Approach to Software Testing:**

The software engineering process can be viewed as a spiral. Initially system engineering defines the role of software and leads to software requirements analysis where the information domain , functions , behavior , performance , constraints and validation criteria for software are established. moving inward along the spiral , we come to design and finally to coding . To develop computer software we spiral in along streamlines that decreases the level of abstraction on each item.

A Strategy for software testing may also be viewed in the context of the spiral. Unit testing begins at the vertex of the spiral and concentrates on each unit of the software as implemented in source code. Testing will progress by moving outward along the spiral to integration testing , where the focus on the design and the concentration of the software architecture. Talking another turn on outward on the spiral we encounter validation testing where requirements established as part of software requirements analysis are validated against the software that has been constructed . Finally we arrive at system testing , where the software and other system elements are tested as a whole .

UNUNI

UNIT TESTING

MODULE

SUB-SYSTEM

**Component**

SYSTEM TESTING

**Integration Testing**

ACCEPTANCE

**User Testing**

**Different Levels of Testing**

Client Needs Acceptance Testing

Requirements System Testing

Design Integration Testing

Code Unit Testing

Testing is the process of finding difference between the expected behavior specified by system models and the observed behavior of the implemented system.

**8.2 Testing Activities**

Different levels of testing are used in the testing process , each level of testing aims to test different aspects of the system. the basic levels are:

Unit testing

Integration testing

System testing

Acceptance testing

**Unit Testing**

Unit testing focuses on the building blocks of the software system, that is, objects and sub system . There are three motivations behind focusing on components. First, unit testing reduces the complexity of the overall tests activities, allowing us to focus on smaller units of the system. Second , unit testing makes it easier to pinpoint and correct faults given that few components are involved in this test . Third , Unit testing allows parallelism in the testing activities , that is each component can be tested independently of one another . Hence the goal is to test the internal logic of the module.

**Integration Testing**

In the integration testing, many test modules are combined into sub systems , which are then tested . The goal here is to see if the modules can be integrated properly, the emphasis being on testing module interaction.

After structural testing and functional testing we get error free modules. These modules are to be integrated to get the required results of the system. After checking a module, another module is tested and is integrated with the previous module. After the integration, the test cases are generated and the results are tested.

**System Testing**

In system testing the entire software is tested . The reference document for this process is the requirement document and the goal is to see whether the software meets its requirements. The system was tested for various test cases with various inputs.

**Acceptance Testing**

Acceptance testing is sometimes performed with realistic data of the client to demonstrate that the software is working satisfactory. Testing here focus on the external behavior of the system , the internal logic of the program is not emphasized . In acceptance testing the system is tested for various inputs.

**8.3 Types of Testing**

1. Black box or functional testing
2. White box testing or structural testing

**Black box testing**

This method is used when knowledge of the specified function that a product has been designed to perform is known . The concept of black box is used to represent a system whose inside workings are not available to inspection . In a black box the test item is a "Black" , since its logic is unknown , all that is known is what goes in and what comes out , or the input and output.

Black box testing attempts to find errors in the following categories:

Incorrect or missing functions

Interface errors

Errors in data structure

Performance errors

Initialization and termination errors

As shown in the following figure of Black box testing , we are not thinking of the internal workings , just we think about

What is the output to our system?

What is the output for given input to our system?

**?**

Input Output

The Black box is an imaginary box that hides its internal workings

**White box testing**

White box testing is concerned with testing the implementation of the program. the intent of structural is not to exercise all the inputs or outputs but to exercise the different programming and data structure used in the program. Thus structural testing aims to achieve test cases that will force the desire coverage of different structures . Two types of path testing are statement testing coverage and branch testing coverage.

**INTERNAL WORKING**

Input Output

The White Box testing strategy , the internal workings

**8.4 Test Plan**

Testing process starts with a test plan. This plan identifies all the testing related activities that must be performed and specifies the schedules , allocates the resources , and specified guidelines for testing . During the testing of the unit the specified test cases are executed and the actual result compared with expected output. The final output of the testing phase is the test report and the error report.

**Test Data:**

Here all test cases that are used for the system testing are specified. The goal is to test the different functional requirements specified in Software Requirements Specifications (SRS) document.

**Unit Testing:**

Each individual module has been tested against the requirement with some test data.

**Test Report:**

The module is working properly provided the user has to enter information. All data entry forms have tested with specified test cases and all data entry forms are working properly.

**Error Report:**

If the user does not enter data in specified order then the user will be prompted with error messages. Error handling was done to handle the expected and unexpected errors.

**8.7 Test cases**

A Test case is a set of input data and expected results that exercises a component with the purpose of causing failure and detecting faults . test case is an explicit set of instructions designed to detect a particular class of defect in a software system , by bringing about a failure . A Test case can give rise to many tests.

**8.3 Result Analysis**

**9. Conclusion and Future Enhancements**

**9.1 Conclusion:**

In this paper, we design a secure data sharing scheme, Mona, for dynamic groups in an untrusted cloud. In Mona, a user is able to share data with others in the group without revealing identity privacy to the cloud. Additionally, Mona supports efficient user revocation and new user joining.

More specially, efficient user revocation can be achieved through a public revocation list without updating the private keys of the remaining users, and new users can directly decrypt files stored in the cloud before their participation. Moreover, the storage overhead and the encryption computation cost are constant.

**9.2 Scope for Future Enhancements**

Extensive analyses show that our proposed scheme satisfies the desired security requirements and guarantees efficiency as well.

REFERENCES

[1] M. Armbrust, A. Fox, R. Griffith, A.D. Joseph, R.H. Katz, A. Konwinski, G. Lee, D.A. Patterson, A. Rabkin, I. Stoica, and M. Zaharia, “A View of Cloud Computing,” Comm. ACM, vol. 53, no. 4, pp. 50-58, Apr. 2010.

[2] S. Kamara and K. Lauter, “Cryptographic Cloud Storage,” Proc. Int’l Conf. Financial Cryptography and Data Security (FC), pp. 136- 149, Jan. 2010.

[3] S. Yu, C. Wang, K. Ren, and W. Lou, “Achieving Secure, Scalable, and Fine-Grained Data Access Control in Cloud Computing,” Proc. IEEE INFOCOM, pp. 534-542, 2010.

[4] M. Kallahalla, E. Riedel, R. Swaminathan, Q. Wang, and K. Fu, “Plutus: Scalable Secure File Sharing on Untrusted Storage,” Proc. USENIX Conf. File and Storage Technologies, pp. 29-42, 2003.

[5] E. Goh, H. Shacham, N. Modadugu, and D. Boneh, “Sirius: Securing Remote Untrusted Storage,” Proc. Network and Distributed Systems Security Symp. (NDSS), pp. 131-145, 2003.

[6] G. Ateniese, K. Fu, M. Green, and S. Hohenberger, “Improved Proxy Re-Encryption Schemes with Applications to Secure Distributed Storage,” Proc. Network and Distributed Systems Security Symp. (NDSS), pp. 29-43, 2005.

[7] R. Lu, X. Lin, X. Liang, and X. Shen, “Secure Provenance: The Essential of Bread and Butter of Data Forensics in Cloud Computing,” Proc. ACM Symp. Information, Computer and Comm. Security, pp. 282-292, 2010.

[8] B. Waters, “Ciphertext-Policy Attribute-Based Encryption: An Expressive, Efficient, and Provably Secure Realization,” Proc. Int’l Conf. Practice and Theory in Public Key Cryptography Conf. Public Key Cryptography, http://eprint.iacr.org/2008/290.pdf, 2008.

[9] V. Goyal, O. Pandey, A. Sahai, and B. Waters, “Attribute-Based Encryption for Fine-Grained Access Control of Encrypted

Data,” Proc. ACM Conf. Computer and Comm. Security (CCS), pp. 89-98, 2006.

[10] D. Naor, M. Naor, and J.B. Lotspiech, “Revocation and Tracing Schemes for Stateless Receivers,” Proc. Ann. Int’l Cryptology Conf. Advances in Cryptology (CRYPTO), pp. 41-62, 2001.

[11] D. Boneh and M. Franklin, “Identity-Based Encryption from the Weil Pairing,” Proc. Int’l Cryptology Conf. Advances in Cryptology (CRYPTO), pp. 213-229, 2001.

[12] D. Boneh, X. Boyen, and H. Shacham, “Short Group Signature,” Proc. Int’l Cryptology Conf. Advances in Cryptology (CRYPTO), pp. 41-55, 2004.

[13] D. Boneh, X. Boyen, and E. Goh, “Hierarchical Identity Based Encryption with Constant Size Ciphertext,” Proc. Ann. Int’l Conf. Theory and Applications of Cryptographic Techniques (EUROCRYPT), pp. 440-456, 2005.

[14] C. Delerablee, P. Paillier, and D. Pointcheval, “Fully Collusion Secure Dynamic Broadcast Encryption with Constant-Size Ciphertexts or Decryption Keys,” Proc. First Int’l Conf. Pairing-Based Cryptography, pp. 39-59, 2007.

[15] D. Chaum and E. van Heyst, “Group Signatures,” Proc. Int’l Conf. Theory and Applications of Cryptographic Techniques (EUROCRYPT), pp. 257-265, 1991.

**REFERENCES:**

**Appendix** Login Code :

String loginname = jTextField1.getText();

String password = jPasswordField1.~~getText~~();

String usertype = jComboBox1.getSelectedItem().toString();

JPanel jpanel = **new** JPanel();

**if** (loginname.equals("")) {

JOptionPane.*showMessageDialog*(jpanel, "Please Enter LoginName");

}

**if** (password.equals("")) {

JOptionPane.*showMessageDialog*(jpanel, "Please Enter Password");

}

**try** {

**if** (!loginname.equals("") && !password.equals("")

&& !usertype.equals("")) {

Users users = registrationDAO.loginCheck(**new** Users(loginname,

password, usertype));

**if** (users != **null**) {

**if** (users.getLogintype().equalsIgnoreCase("owner")) {

OwnerForm ownerForm = **new** OwnerForm();

ownerForm.setVisible(**true**);

} **else** **if** (users.getLogintype().equalsIgnoreCase("user")) {

Users taskrequest = **new** Users();

taskrequest.setUserid(users.getUserid());

taskrequest.setLoginid(users.getLoginid());

taskrequest

.setRegistereddate(users.getRegistereddate());

taskrequest.setLogintype(users.getLogintype());

taskrequest.setLocation(users.getLocation());

taskrequest.setMailid(users.getMailid());

UsersForm usersForm = **new** UsersForm(treeMap,

taskrequest);

usersForm.setVisible(**true**);

} **else** {

PublicCloudOwnerForm publicCloudOwnerForm = **new** PublicCloudOwnerForm(

treeMap, users);

publicCloudOwnerForm.setVisible(**true**);

}

} **else** {

JOptionPane.*showMessageDialog*(jpanel,

"Wrong loginid and password");

}

}

} **catch** (Exception e) {

System.*out*.println(e);

JOptionPane.*showMessageDialog*(jpanel,

"INVALID USER PLEASE TRY AGAIN");

}

File Upload and Downlaod :

String status = registrationDAO.getFileStatus(jList1.getSelectedValue()

.toString());

**if** (status.equals("false")) {

JOptionPane.*showMessageDialog*(jPanel,

"wait for Cloud Server provider permission");

} **else** {

String privatekey = JOptionPane

.*showInputDialog*("Please Enter Private Key");

**if** (privatekey != **null**) {

String existedkey = registrationDAO.getKey(jList1

.getSelectedValue().toString());

**if** (existedkey.trim().equals(privatekey.trim())) {

JOptionPane.*showMessageDialog*(**null**, "Key Matched",

"Private Key", 1);

Users users = (Users) treeMap.get(count);

**if** (!(users.getDownloadtime() > users.getMintime())) {

**try** {

String downloadfile = registrationDAO

.downloadFile(jList1.getSelectedValue()

.toString());

**if** (!downloadfile.equals("")) {

JFileChooser c = **new** JFileChooser();

**int** x = c.showSaveDialog(**null**);

**if** (x == JFileChooser.*APPROVE\_OPTION*) {

**try** {

String saveFilename = **null**;

String filename = **new** File(downloadfile)

.getName().toString();

System.*out*.println(filename);

String[] token = filename.split("\\.");

**if** (c.getSelectedFile() != **null**) {

**try** {

saveFilename = c

.getSelectedFile()

+ "." + token[1];

FileInputStream fileInputStream = **new** FileInputStream(

**new** File(downloadfile));

FileOutputStream fileOutputStream = **new** FileOutputStream(

saveFilename);

**int** data;

**while** ((data = fileInputStream

.read()) != -1) {

fileOutputStream

.write(data);

// fileOutputStream.write((int)

// data);

}

} **catch** (Exception e) {

// **TODO**: handle exception

}

} **else** {

saveFilename = **new** File(

downloadfile).getName()

.toString();

}

System.*out*.println(saveFilename);

} **catch** (Exception e) {

e.printStackTrace();

}

}

}

} **catch** (Exception e) {

System.*out*.println(e);

}

} **else** {

JOptionPane.*showMessageDialog*(**null**,

"Time is not sufficient left time to download :"

+ users.getDifftime() + "secs",

"Private Key", 1);

}

}

**else**

JOptionPane.*showMessageDialog*(**null**, "Key Not Matched",

"Private Key", 1);

}

**USER SERVER NODE:**

taskrequest.setFilename(jList1.getSelectedValue().toString());

Users users = registrationDAO.getTaskProcessingDetails(jList1

.getSelectedValue().toString());

**int** filelength = registrationDAO.getFileLength(jList1

.getSelectedValue().toString());

**int** status = registrationDAO.getFileStatusChange(jList1

.getSelectedValue().toString(), "false");

taskrequest.setFilesize(filelength);

taskrequest.setDownloadtime((**int**) filelength / 100);

taskrequest.setBandwidth(users.getBandwidth());

taskrequest.setDownloadcost(users.getVmcapacity()

/ users.getDownloadcost());

taskrequest.setCloudservername(users.getCloudservername());

taskrequest.setVirtuvalmachine(users.getVirtuvalmachine());

treeMap.put(count, taskrequest);

JOptionPane.*showMessageDialog*(jPanel, "Request sent to Cloud Server");

SENT REQUEST TO CLOUD OWNER:

String status = registrationDAO.getFileStatus(jList1.getSelectedValue()

.toString());

**if** (status.equals("false")) {

JOptionPane.*showMessageDialog*(jPanel,

"wait for Cloud Server provider permission");

} **else** {

String privatekey = JOptionPane

.*showInputDialog*("Please Enter Private Key");

**if** (privatekey != **null**) {

String existedkey = registrationDAO.getKey(jList1

.getSelectedValue().toString());

**if** (existedkey.trim().equals(privatekey.trim())) {

JOptionPane.*showMessageDialog*(**null**, "Key Matched",

"Private Key", 1);

Users users = (Users) treeMap.get(count);

**if** (!(users.getDownloadtime() > users.getMintime())) {

**try** {

String downloadfile = registrationDAO

.downloadFile(jList1.getSelectedValue()

.toString());

**if** (!downloadfile.equals("")) {

JFileChooser c = **new** JFileChooser();

**int** x = c.showSaveDialog(**null**);

**if** (x == JFileChooser.*APPROVE\_OPTION*) {

**try** {

String saveFilename = **null**;

String filename = **new** File(downloadfile)

.getName().toString();

System.*out*.println(filename);

String[] token = filename.split("\\.");

**if** (c.getSelectedFile() != **null**) {

**try** {

saveFilename = c

.getSelectedFile()

+ "." + token[1];

FileInputStream fileInputStream = **new** FileInputStream(

**new** File(downloadfile));

FileOutputStream fileOutputStream = **new** FileOutputStream(

saveFilename);

**int** data;

**while** ((data = fileInputStream

.read()) != -1) {

fileOutputStream

.write(data);

// fileOutputStream.write((int)

// data);

}

} **catch** (Exception e) {

// **TODO**: handle exception

}

} **else** {

saveFilename = **new** File(

downloadfile).getName()

.toString();

}

System.*out*.println(saveFilename);

} **catch** (Exception e) {

e.printStackTrace();

}

}

}

} **catch** (Exception e) {

System.*out*.println(e);

}

} **else** {

JOptionPane.*showMessageDialog*(**null**,

"Time is not sufficient left time to download :"

+ users.getDifftime() + "secs",

"Private Key", 1);

}

}

**else**

JOptionPane.*showMessageDialog*(**null**, "Key Not Matched",

"Private Key", 1);

}

}

Key Generation Graph :

**try** {

dataset = **new** XYSeriesCollection();

**int** k = 0;

**for** (**int** i = 1; i <= timemap.size(); i++) {

**final** XYSeries series = **new** XYSeries(timemap.get(i)

.getLoginid()

+ "--task");

// final XYSeries series = new XYSeries((i+1) + "--task");

k = 0;

**for** (**int** j = 0; j < timemap.get(i).getDownloadtime(); j++) {

k = k + 3;

series.add(i + 1, j);

}

dataset.addSeries(series);

}

**final** XYSeries series1 = **new** XYSeries("MIN TIME");

// int k = 0;

**for** (**int** j = 0; j < 10; j++) {

series1.add(j, mintime);

}

dataset.addSeries(series1);

} **catch** (Exception e) {

System.*out*.println(e);

}

**return** dataset;

Public Cloud Owner Form Operations :

**private** **void** jTable2MouseExited(java.awt.event.MouseEvent evt) {

String cloudServerName = jComboBox1.getSelectedItem().toString();

Object obj[][] = registrationDAO.viewVirtuvalMachines(cloudServerName);

Object string[] = registrationDAO.viewVirtuvalMachineMetaData();

jTable2.setModel(**new** javax.swing.table.DefaultTableModel(obj, string));

jScrollPane2.setViewportView(jTable2);

jButton3.setText("Create Server Nodes");

String serverNames = registrationDAO

.viewVirtuvalMachineNames(cloudServerName);

jButton4.setText("Delete Server Nodes");

jComboBox2.setModel(**new** javax.swing.DefaultComboBoxModel(serverNames

.split(",")));

}

**private** **void** jTable1MouseExited(java.awt.event.MouseEvent evt) {

Object obj[][] = registrationDAO.viewCloudServers();

Object string[] = registrationDAO.viewCloudServerMetaData();

jTable1.setModel(**new** javax.swing.table.DefaultTableModel(obj, string));

jScrollPane1.setViewportView(jTable1);

String cloudServerNames[] = registrationDAO.viewCloudServerNames()

.split(",");

jComboBox1.setModel(**new** javax.swing.DefaultComboBoxModel(

cloudServerNames));

}

**private** **void** jTable1MouseMoved(java.awt.event.MouseEvent evt) {

// **TODO** add your handling code here:

}

**private** **void** jButton8ActionPerformed(java.awt.event.ActionEvent evt) {

JPanel jPanel = **new** JPanel();

String status = JOptionPane

.*showInputDialog*("Please Enter Status Only Idle (or) Normal (or) Overload");

**if** (status.toLowerCase().equals("idle")

|| status.toLowerCase().equals("normal")

|| status.toLowerCase().equals("overload")) {

**int** flag = 0;

**try** {

flag = registrationDAO

.changeServerNodeStatus(status.toLowerCase(),

jComboBox2.getSelectedItem().toString());

} **catch** (Exception e) {

System.*out*.println(e);

}

**if** (flag > 0) {

JOptionPane.*showMessageDialog*(jPanel,

"Status Changed SuccessFully...");

}

repaint();

} **else** {

JOptionPane.*showMessageDialog*(jPanel, "Please Enter Proper String");

}

}

**private** **void** jButton4ActionPerformed(java.awt.event.ActionEvent evt) {

JPanel jPanel = **new** JPanel();

**boolean** flag = **false**;

**try** {

flag = registrationDAO.deleteServerNodes(jComboBox2

.getSelectedItem().toString());

} **catch** (Exception e) {

System.*out*.println(e);

}

**if** (flag) {

JOptionPane.*showMessageDialog*(jPanel,

"Server Node Deleted SuccessFully...");

} **else** {

JOptionPane.*showMessageDialog*(jPanel,

"Server Node Deleted SuccessFully...");

}

}

**private** **void** jButton9ActionPerformed(java.awt.event.ActionEvent evt) {

JPanel jPanel = **new** JPanel();

String status = JOptionPane

.*showInputDialog*("Please Enter Statsu Only Idle (or) Normal (or) Overload");

**if** (status.toLowerCase().equals("idle")

|| status.toLowerCase().equals("normal")

|| status.toLowerCase().equals("overload")) {

**int** flag = 0;

**try** {

flag = registrationDAO.changeServerStatus(status.toLowerCase(),

jComboBox1.getSelectedItem().toString());

} **catch** (Exception e) {

System.*out*.println(e);

}

**if** (flag > 0) {

JOptionPane.*showMessageDialog*(jPanel,

"Status Changed SuccessFully...");

}

repaint();

} **else** {

JOptionPane.*showMessageDialog*(jPanel, "Please Enter Proper String");

}

}

**private** **void** jButton3ActionPerformed(java.awt.event.ActionEvent evt) {

CreateVirtuvalMachineForm createNodeForm = **new** CreateVirtuvalMachineForm(

jComboBox1.getSelectedItem().toString());

createNodeForm.setVisible(**true**);

}

**protected** **void** jButton2ActionPerformed(ActionEvent evt) {

String cloudServerName = jComboBox1.getSelectedItem().toString();

Object obj[][] = registrationDAO.viewVirtuvalMachines(cloudServerName);

Object string[] = registrationDAO.viewVirtuvalMachineMetaData();

jTable2.setModel(**new** javax.swing.table.DefaultTableModel(obj, string));

jScrollPane2.setViewportView(jTable2);

jButton3.setText("Create Server Nodes");

String serverNames = registrationDAO

.viewVirtuvalMachineNames(cloudServerName);

jButton4.setText("Delete Server Nodes");

jComboBox2.setModel(**new** javax.swing.DefaultComboBoxModel(serverNames

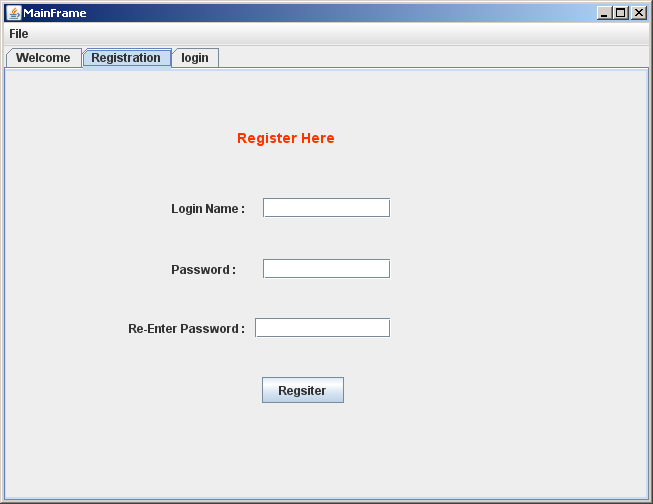
.split(",")));

}

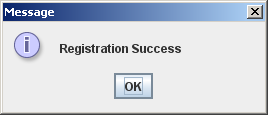
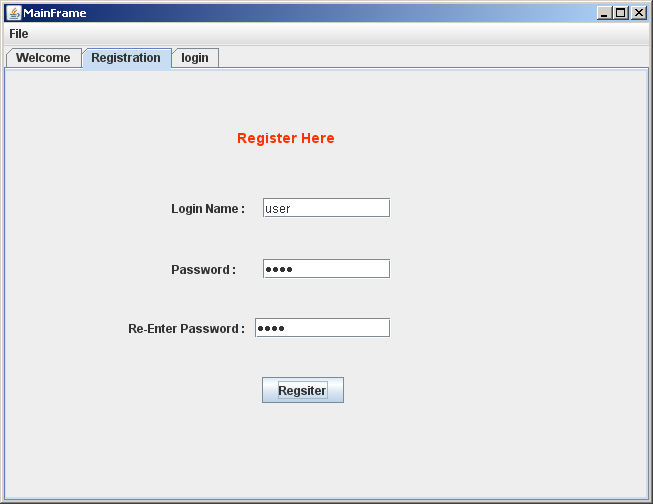
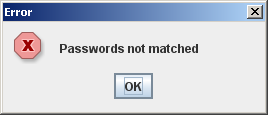
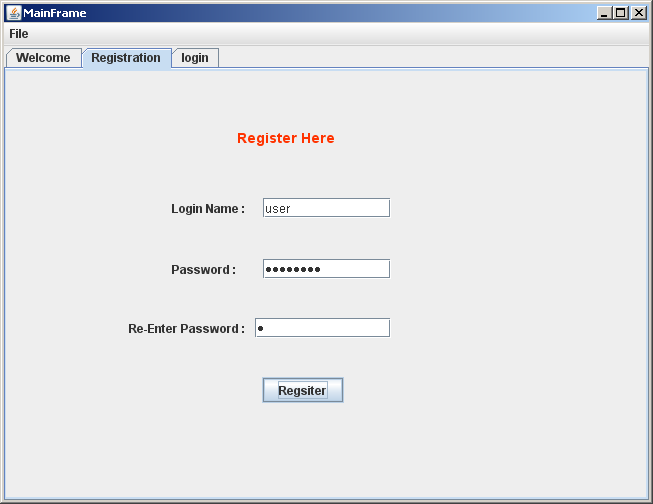
Home page



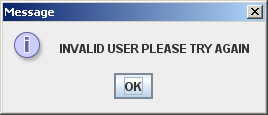
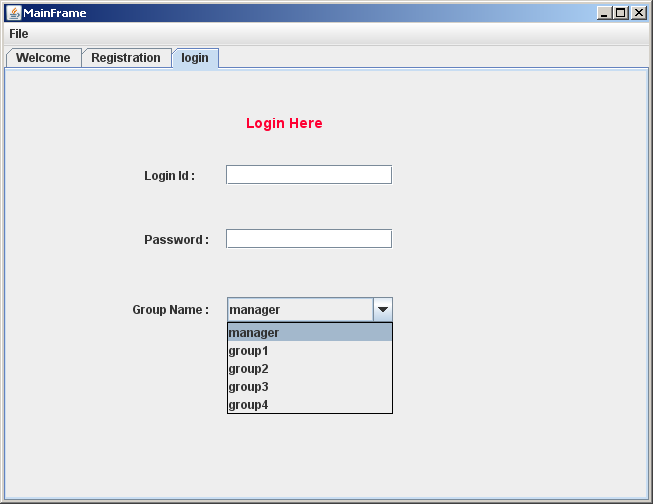
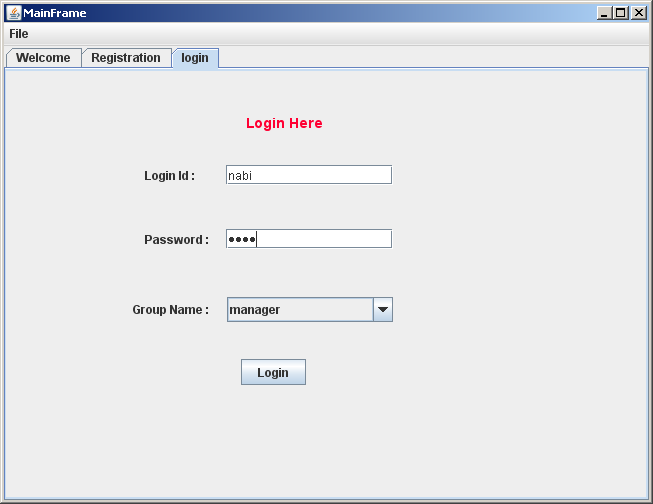
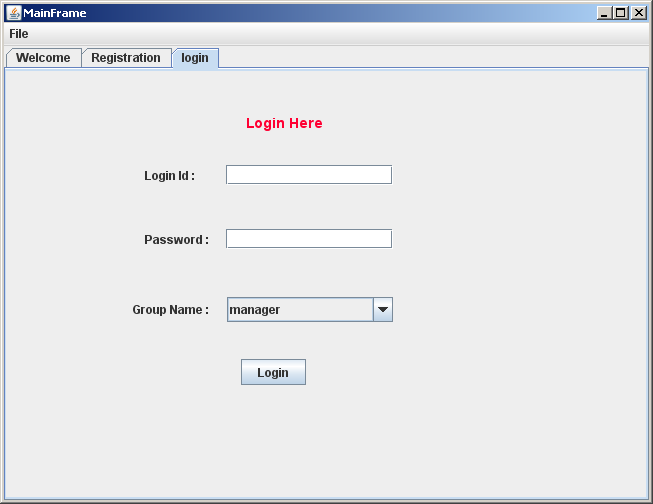
Registration page



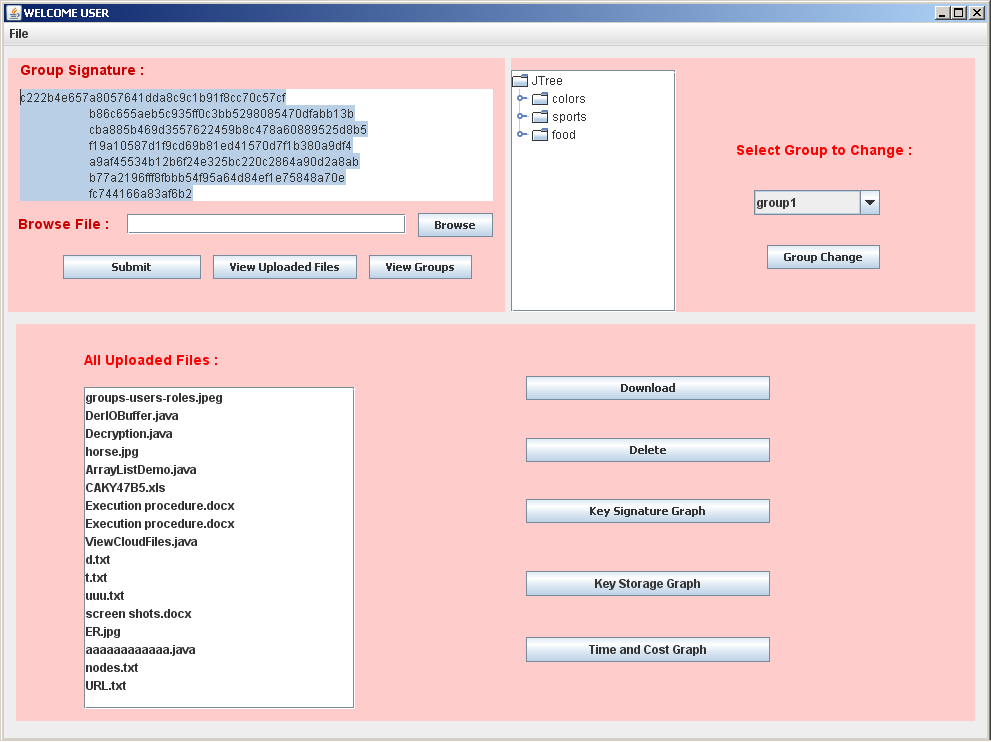
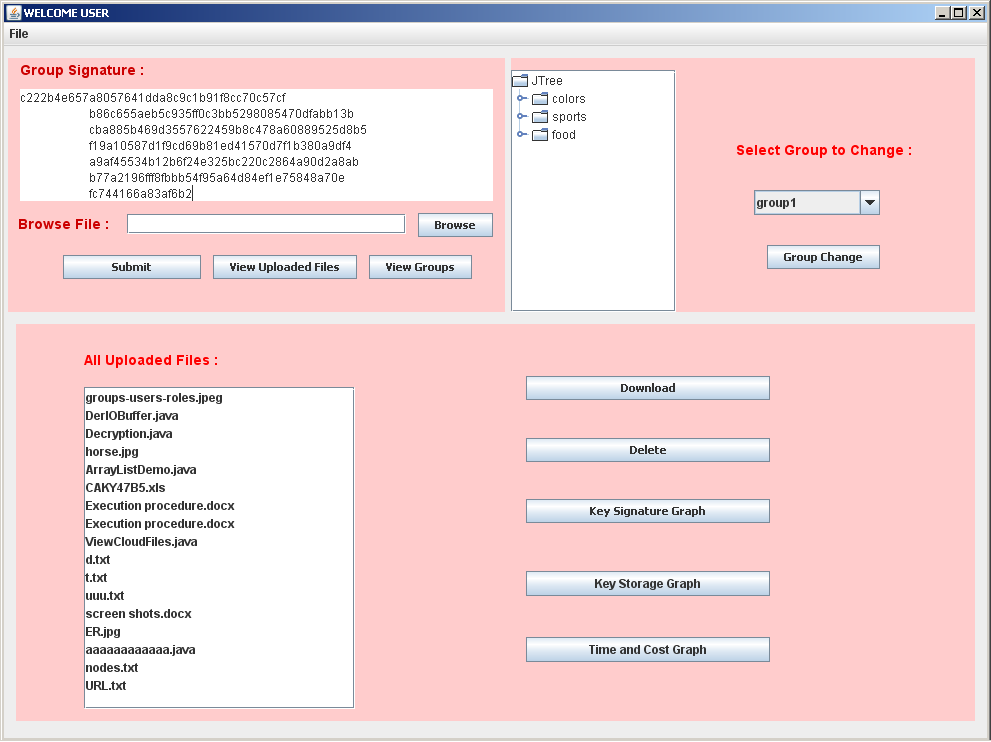
Password not matched :



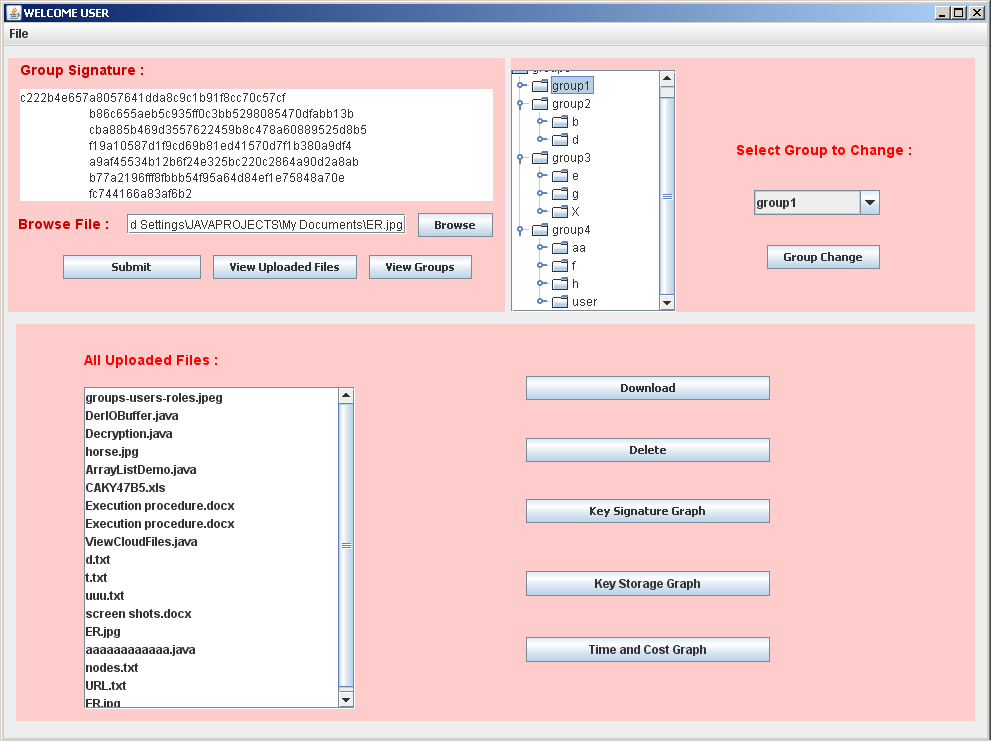
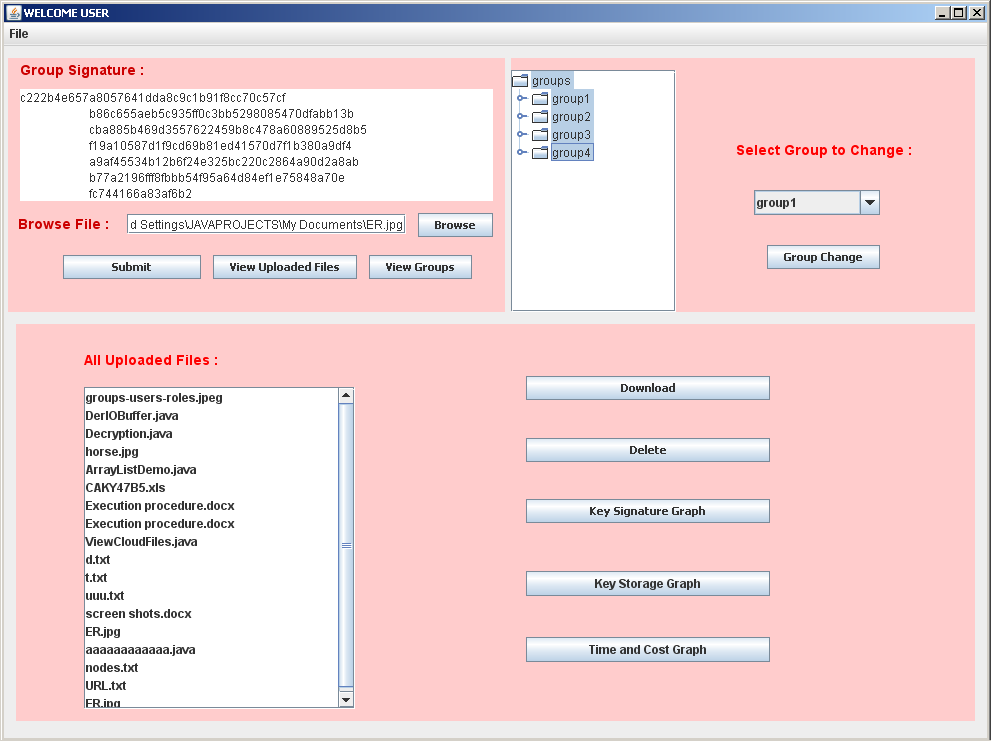
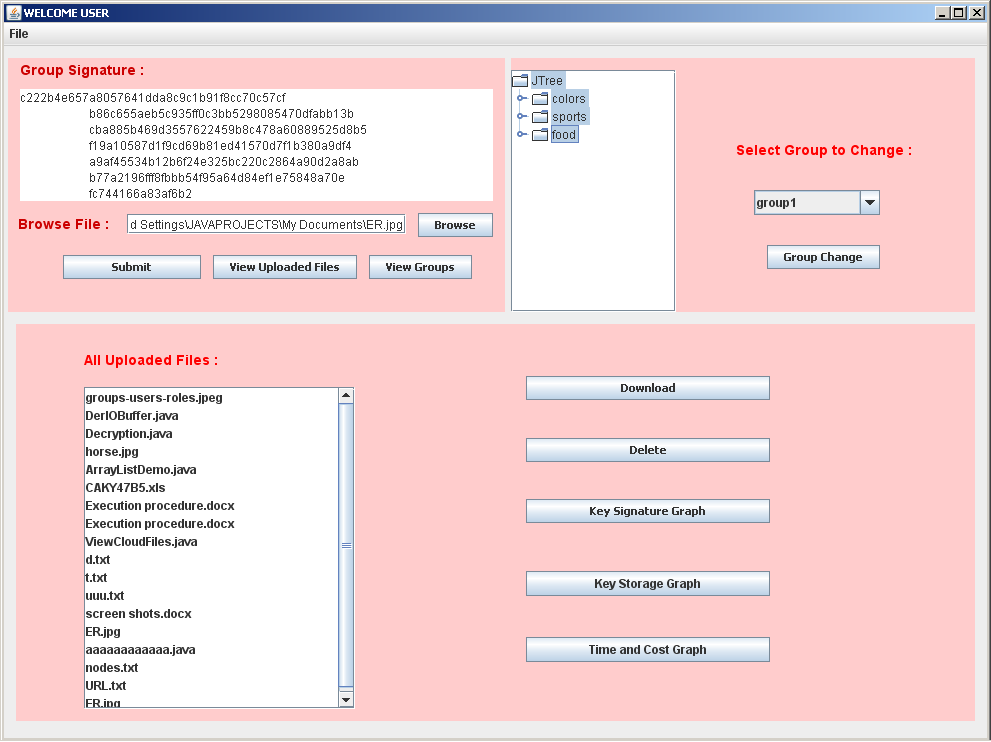
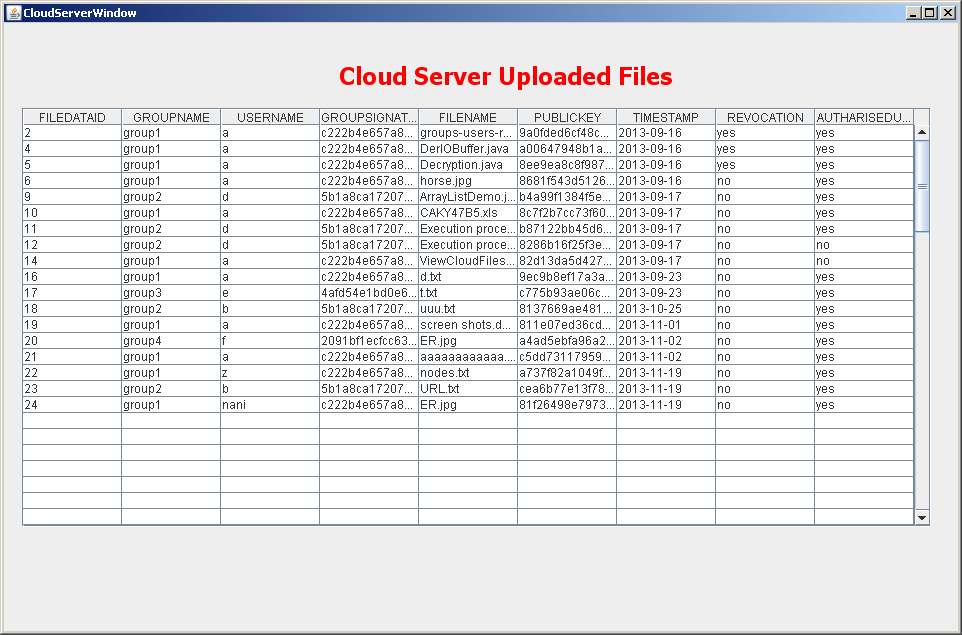
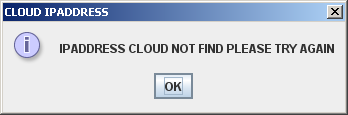
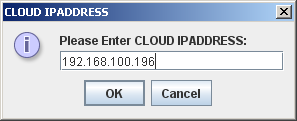
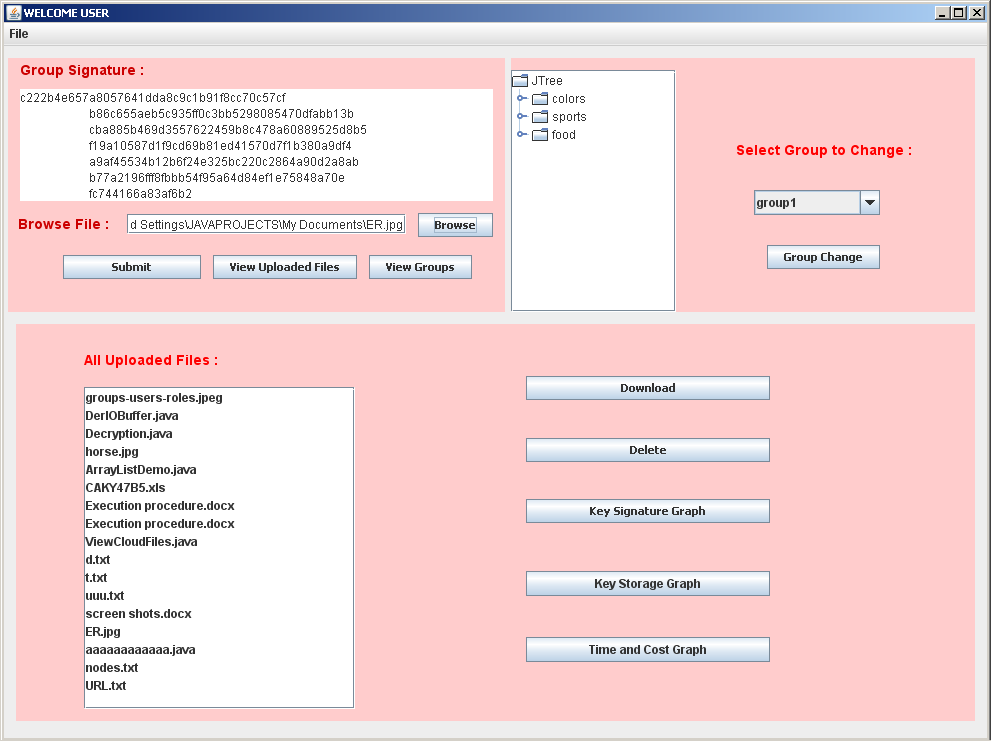
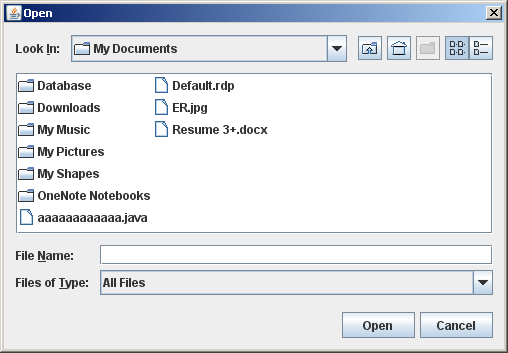
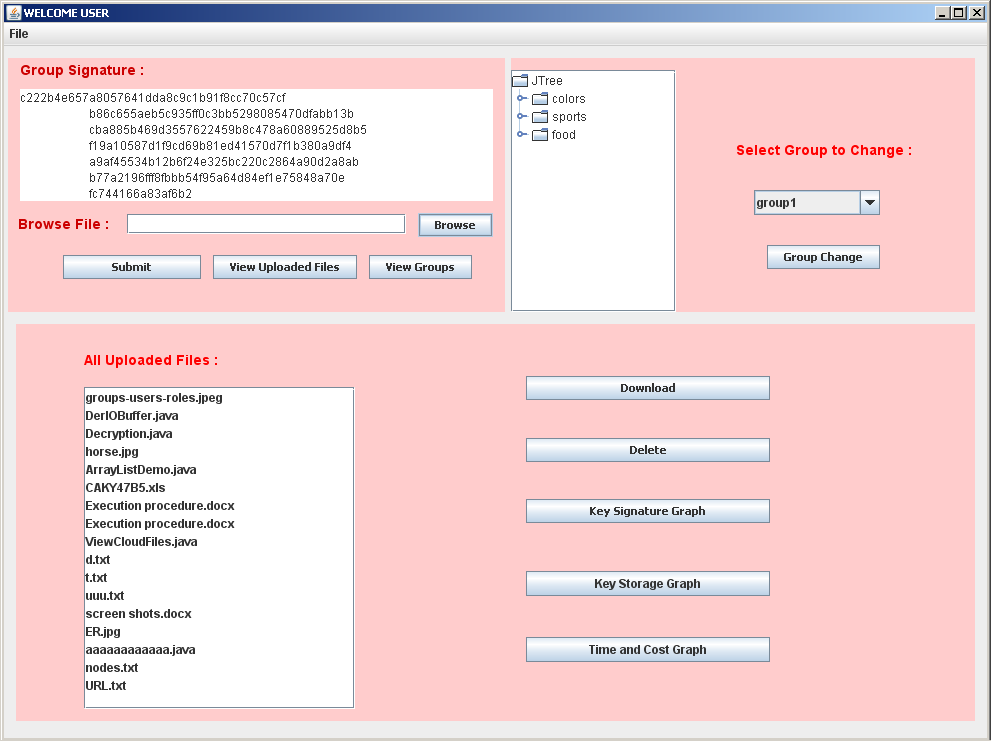
Login page



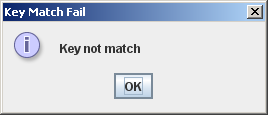
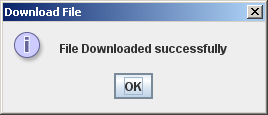
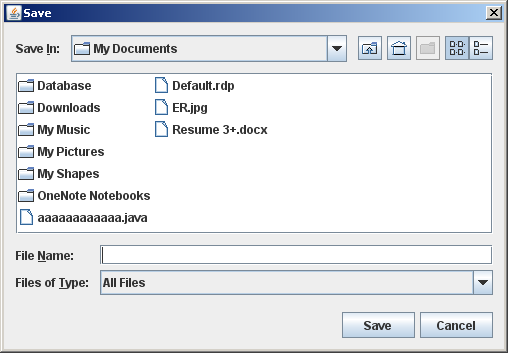
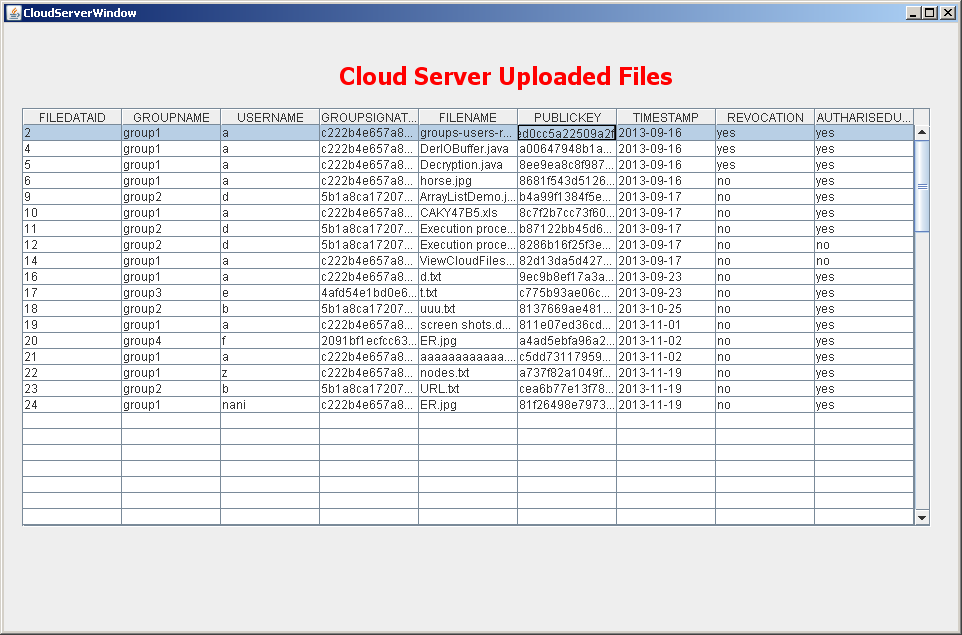
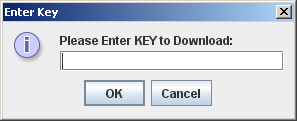
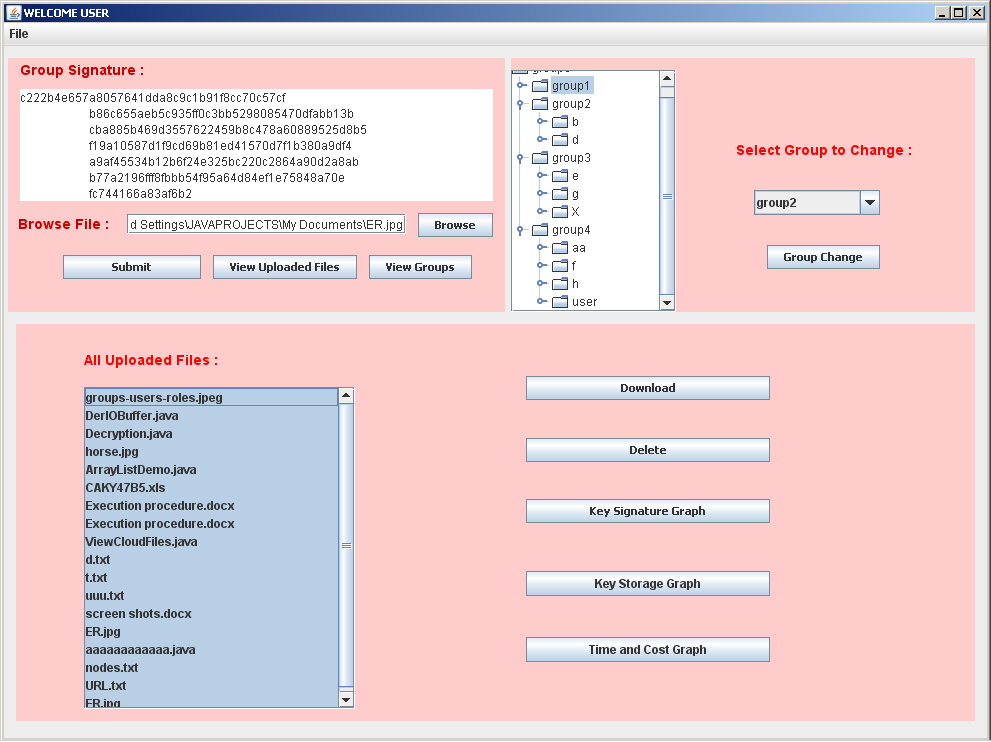
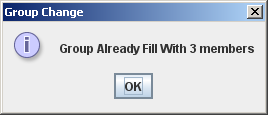
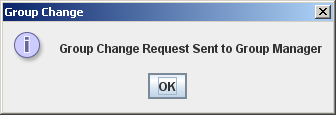
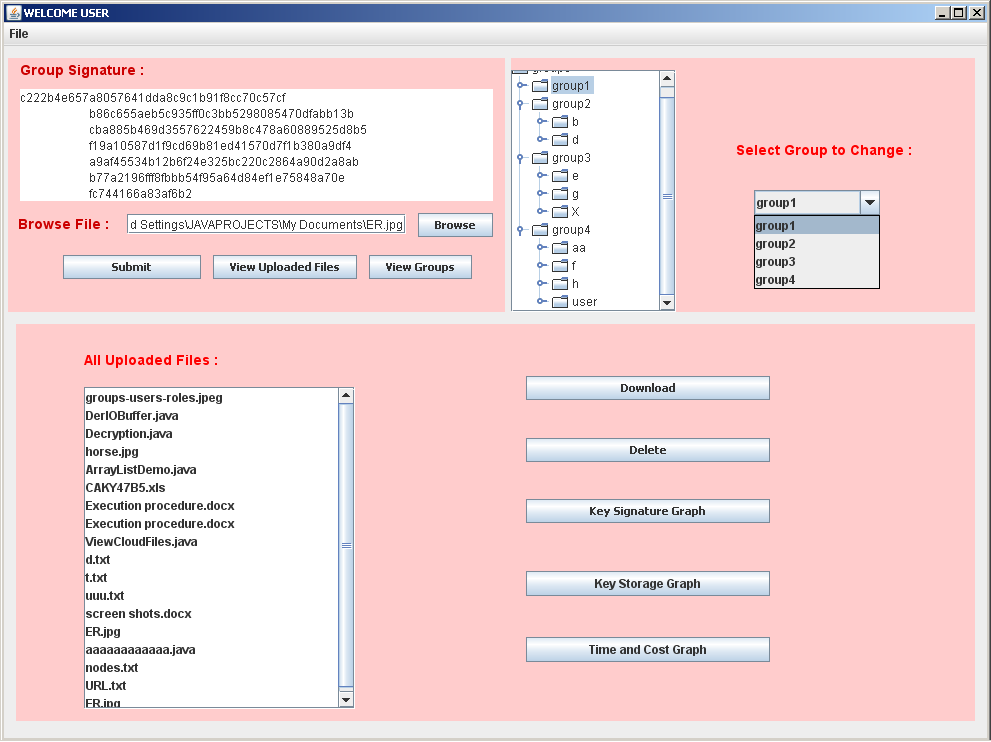
User Home page



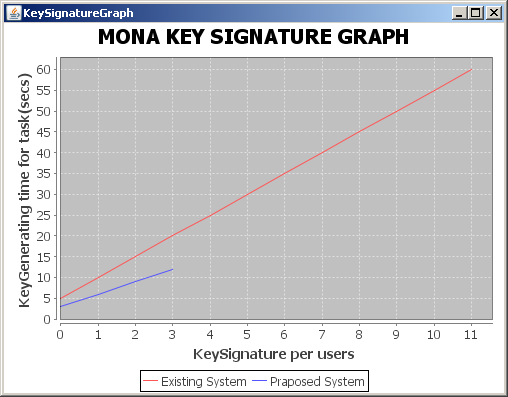
Upload file :



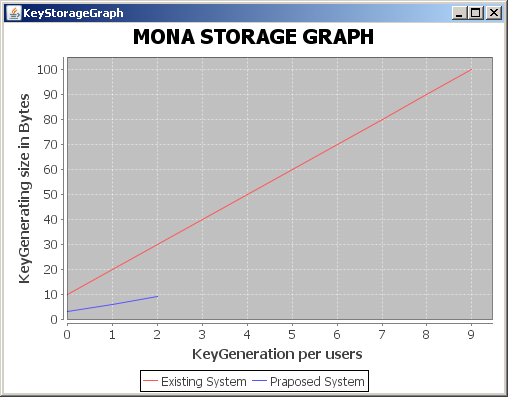
Group Changed Request :



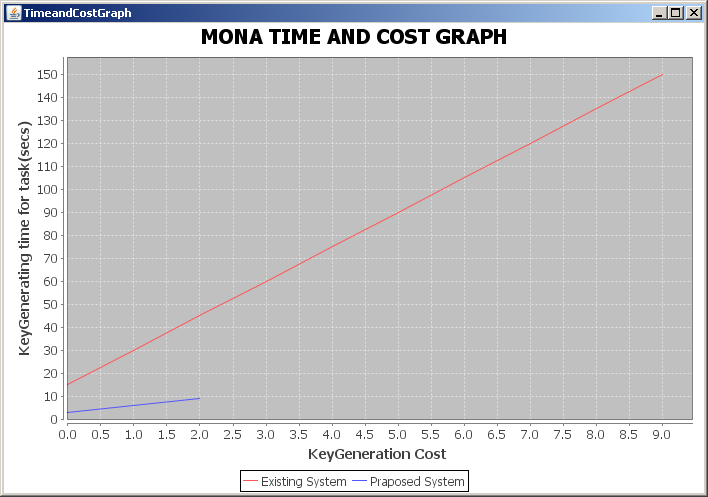
Signature graph :



Mona Storage Graph



Mona time and cost Graph



Manager Window :

