

**ANALYSIS OF CLOUD COMPUTING SECURITY CHALLENGES
AND THREATS FOR RESOLVING DATA BREACH ISSUES**

Submitted in partial fulfillment of the requirements for the award of
Bachelor of Engineering degree in Computer Science and Engineering

By

CHOKKAKULA CHANDRA MOULI (Reg No - 40110280)

SEERAMREDDI PAVAN KALYAN (Reg No - 40111162)



**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
SCHOOL OF COMPUTING**

SATHYABAMA

**INSTITUTE OF SCIENCE AND TECHNOLOGY
(DEEMED TO BE UNIVERSITY)**

CATEGORY -1 University by UGC

**Accredited with Grade “A++” by NAAC | 12B Status by UGC | Approved by AICTE
JEPPIAAR NAGAR, RAJIV GANDHI SALAI, CHENNAI – 600119**

APRIL - 2024

SATHYABAMA

INSTITUTE OF SCIENCE AND TECHNOLOGY

(DEEMED TO BE UNIVERSITY)

Category - I University by UGC

Accredited "A++" by NAAC | Approved by AICTE

www.sathyabama.ac.in

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

BONAFIDE CERTIFICATE

This is to certify that this Project Report is the bonafide work of CHOKKAKULA CHANDRA MOULI (40110280) and SEERAMREDDI PAVAN KALYAN (40111162) who carried out the Project entitled "ANALYSIS OF CLOUD COMPUTING CHALLENGES AND THREATS FOR RESOLVING DATA BREACH ISSUES" under my supervision from November 2023 to April 2024.

Internal Guide

Dr. T. JUDGI M.E., Ph.D.

Head of the Department
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
Dr. SAKSHMANAN M.E., Ph.D.
SATHYABAMA
INSTITUTE OF SCIENCE AND TECHNOLOGY
(DEEMED TO BE UNIVERSITY)
Jeppiaar Nagar, Rajiv Gandhi Salai,
Chennai - 600 119.

Submitted for Viva voce Examination held on

13/04/2024

Dr. C. Henditha
Internal Examiner

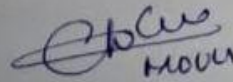
P. Arul
External Examiner

DECLARATION

I, CHOKKAKULA CHANDRA MOULI (40110280), hereby declare that the Project Report entitled "ANALYSIS OF CLOUD COMPUTING CHALLENGES AND THREATS FOR RESOLVING DATA BREACH ISSUES" done by me under the guidance of Dr. T. JUDGI M.E., Ph.D., is submitted in partial fulfillment of the requirements for the award of Bachelor of Engineering degree in Computer Science and Engineering.

DATE: 13/04/2024

PLACE: Chennai

A handwritten signature in blue ink, appearing to read 'Chokkula Chandra Mouli', with the name 'MOULI' written in capital letters below the signature.

SIGNATURE OF CANDIDATE

ACKNOWLEDGEMENT

I am pleased to acknowledge my sincere thanks to **Board of Management** of **SATHYABAMA INSTITUTE OF SCIENCE AND TECHNOLOGY** for their kind encouragement in doing this project and for completing it successfully. I am grateful to them.

I convey my thanks to **Dr. T. Sasikala M.E., Ph.D., Dean**, School of Computing, **Dr. L. Lakshmanan M.E., Ph.D.**, Head of the Department of Computer Science and Engineering for providing me necessary support and details at the right time during the progressive reviews.

I would like to express my sincere and deep sense of gratitude to my Project Guide for her **Dr. T. Judgi M.E., Ph.D.**, valuable guidance, suggestions and constant encouragement paved way for the successful completion of my project work.

I wish to express my thanks to all Teaching and Non-teaching staff members of the **Department of Computer Science and Engineering** who were helpful in many ways for the completion of the project.

ABSTRACT

Cloud computing is considered as one of the most prominent paradigms in the information technology industry, since it can significantly reduce the costs of hardware and software resources in computing infrastructure. At the first sight, by merely storing the shared data as plaintext in the cloud storage and protect them using an appropriate access control would be a nice solution. This is assuming that the cloud is fully trusted for not leaking any information, which is impractical as the cloud is owned by a third party. Therefore, encryption is mandatory, and the shared data will need to be stored as a ciphertext using an appropriate access control. The existing protection in the literature has been explored to allow only legitimate recipients to decrypt the contents stored in the cloud storage, but unfortunately, no existing work deals with issues raised due to the presence of malicious data publishers. Malicious data publishers construct data following the given policy, but the ciphertexts can actually be decrypted by unauthorized users without valid keys, or simply, anyone else who is unauthorized. The impact of the involvement of malicious data publishers is detrimental, as it may damage intellectual properties from the corporations. In this work, we present a new direction of research that can cope with the presence of malicious data publishers. We resolve the aforementioned problem by proposing the notion of Sanitizable Access Control System (SACS), which is designed for a secure cloud storage that can also resist against malicious data publishers. We define the threat model and its formal security model, as well as its design and scheme which is based on q -Parallel Bilinear Diffie-Hellman Exponent Assumption. We provide the security proof of our construction as well as its performance analysis. We believe that this work has opened a new area of research which has never been explored before, even though it is very practical. Therefore, this work will enhance the adoption of secure cloud storage in practice. We define the threat model and its formal security model, as well as its design and scheme which is based on q -Parallel Bilinear Diffie-Hellman Exponent Assumption. The impact of the involvement of malicious data publishers is detrimental, as it may damage intellectual properties from the corporations. In this work, we present a new direction of research that can cope with the presence of malicious data publishers.

TABLE OF CONTENTS

Chapter No	TITLE	Page No
	ABSTRACT	v
	LIST OF FIGURES	viii
	LIST OF ABBREVIATIONS	ix
1	INTRODUCTION	1
	1.1 Background and Motivation	2
	1.2 Problem Statement	3
	1.3 Objective and Scope	3
	1.4 Organization of Thesis	4
2	LITERATURE SURVEY	5
	2.1 Overview of Existing Research and Literature	5
	2.2 Inferences from Literature Survey	6
3	REQUIREMENTS ANALYSIS	8
	3.1 Feasibility Studies of the Project	8
	3.1.1 Economic Feasibility	8
	3.1.2 Technical Feasibility	8
	3.1.3 Social Feasibility	9
	3.2 Requirements Specification	10
	3.2.1 Hardware Requirements	10
	3.2.2 Software Requirements	10
4	DESCRIPTION OF PROPOSED SYSTEM	11
	4.1 Selected Methodology or Process Model	11
	4.2 Architecture / Overall Design of Proposed System	11

4.3	Description of Software for Implementation and Testing	12
4.3.1	Types of Tests	12
4.3.2	Unit Testing	14
4.3.3	Integration Testing	14
5	IMPLEMENTATION DETAILS	16
5.1	Development and Deployment Setup	16
5.2	Algorithms	17
5.2.1	Convolutional and Deployment Setup	17
5.2.2	Long Short Term Memory	17
6	RESULTS AND DISCUSSION	19
7	CONCLUSION	22
7.1	Conclusion	22
7.2	Future Work	22
	REFERENCES	23
	APPENDIX	25
	A. SOURCE CODE	25
	B. SCREEN SHOTS	48
	C. RESEARCH PAPER	51
	D. CERTIFICATE	56

LIST OF FIGURES

Fig no.	FIGURE NAME	PAGE NO
4.1	Architecture Diagram	11
5.1	Development and Deployment Setup	16
6.1	Graph for Cloud Share	18
7.1	Sanitizable access control system	48
7.2	Sanitizer Home Snapshot	48
7.3	Cloud Home Snapshot	49
7.4	Authority Home Snapshot	49
7.5.	Authority Login Snapshot	50
7.6	Active Data Publisher Snapshot	50

LIST OF ABBREVIATIONS

ABBREVIATIONS	EXPANSION
ABE	Attribute Based Encryption
DES	Data Encryption Standard
RSA	Rivest Shamir Algorithm
SACS	Sanitizable Access Control Systems
SME	Small and Medium Sized Enterprises

CHAPTER 1

INTRODUCTION

1.1 Introduction

The emergence of cloud storage technology has greatly influenced enterprise operations and the adoption of cloud technology has been one of the biggest changes of the digital age. Cloud storage offers low-cost solutions. Which would be perfect for business, such as Small and medium sized Enterprises (SMEs). The existence of cloud storage enables business corporations to conveniently share their data among their employees. These data are supposed to be used solely by the employees of those corporations, since those may be related to their intellectual properties. At the first sight, by merely storing the data as plaintext in the cloud storage and protecting them by using an appropriate access control would be a sufficient solution. This is based on the assumption that the cloud is fully trusted and will not leak those data, which is impractical, as the cloud is owned by a third party. The existing body of work in the literature make use of the notion of Attribute-based Encryption (ABE) to enable such an unauthorized prevention by protecting the data with an appropriate access policy. Anyone who is equipped with a valid decryption key satisfying the access policy will be able to decrypt the data correctly. It implies that the data will be stored as a ciphertext instead of a plaintext in the cloud. This kind of protection only considers data privacy when data publishers are honest and follow the encryption algorithm. Unfortunately, in practice, some of these employees may be malicious and they intentionally attempt to leak the contents of those data to unauthorized recipients, such as competing business corporations. These malicious employees may even would like to publish some sensitive contents and store them in the cloud, but also enable other unauthorized users to retrieve it- hence constituting a malicious data publisher. It is unfortunate that the approach based on ABE is not sound due to malicious data publishers, who encrypt data in a pernicious way. Here, the malicious data publisher constructs encrypted data following the given policy, but the ciphertexts can actually be decrypted by unauthorized users without valid keys. In practice, malicious data publishers refer to staff or virus- infected computers of a company that intend to leak internal sensitive information. As an example, a malicious data publisher may want to leak new product designs or commercial secrets.

by certain people. The impact of the involvement of malicious data publishers is detrimental. In the above setting, malicious data publishers will construct ciphertexts containing copy righted materials that seemingly adhere the required security access policy set by the organization. Nevertheless, the supposedly encrypted file can actually be decrypted by anyone illegally without any valid decryption key. Therefore, The main goal is to achieve data privacy when data publishers are malicious and they do not follow the encryption algorithm accordingly.

The aim to propose a very practical notion, called Sanitizable Access control system, or simply SACS, which is designed for the cloud storage to resist against malicious data publishers. SACS enables a flexible access control for both data publishers and data receivers. As in ABE, SACS allows any valid receivers who are equipped with private keys satisfying the access policy to decrypt the ciphertext. However, SACS is equipped with sanitizing capability, which prevents malicious data publishers can maliciously generate ciphertexts which can be decrypted by anyone, the sanitizer will transform these ciphertexts into new ciphertexts which will only be decryptable by valid private key holders. Our architecture as well as our scheme to achieve the above concept to build SACS. Furthermore, it will also present an implementation of SACS.

1.1 BACKGROUND AND MOTIVATION

The emergence of cloud storage technology has revolutionized the way enterprises manage and share data, particularly benefiting small and medium-sized enterprises (SMEs) by offering low-cost solutions and convenient data sharing among employees. However, this convenience comes with inherent risks, especially regarding data privacy and security. While conventional encryption methods like Attribute-based Encryption (ABE) address some of these concerns, they fall short in scenarios where employees or systems turn malicious, potentially leaking sensitive information to unauthorized parties. This gap in security necessitates the development of a practical solution capable of protecting data integrity even in the face of malicious actors. Motivated by this imperative, the proposed Sanitizable Access Control System (SACS) aims to fortify cloud storage against such threats by incorporating a flexible access control mechanism with built-in sanitization capabilities. This innovative approach not only allows authorized users to decrypt

data as per the access policy but also safeguards against malicious data publishers who may attempt to subvert encryption protocols. Thus, the background of increasing reliance on cloud storage and the motivation to address its inherent vulnerabilities converge to drive the development of SACS, ensuring robust data privacy and security in the digital age.

1.2 PROBLEM STATEMENT

The problem statement revolves around the challenge of ensuring data privacy and security in cloud storage environments, particularly when dealing with malicious data publishers who intentionally subvert encryption algorithms to leak sensitive information to unauthorized recipients. While traditional approaches like Attribute-based Encryption (ABE) provide some level of protection, they fall short in preventing malicious actors from constructing ciphertexts that can be decrypted by anyone, bypassing access control mechanisms. This poses a significant threat to businesses, especially small and medium-sized enterprises (SMEs), whose intellectual property and confidential data are at risk of exposure. To address this issue, the proposed solution introduces the concept of a Sanitizable Access Control System (SACS), designed specifically to thwart malicious data publishers by sanitizing ciphertexts to ensure they can only be decrypted by authorized parties with valid private keys. The problem statement thus centers on the need for a practical and effective solution to safeguard data privacy in cloud storage environments against malicious insiders and external threats.

1.3 OBJECTIVE AND SCOPE

Objective

The objective of SACS is to combat malicious data publishers in cloud storage, crucial for ensuring data privacy amidst increasing cloud reliance, especially for SMEs. Unlike traditional methods like ABE, SACS offers a flexible access control system with sanitizing capabilities. It enables authorized decryption based on policies while preventing unauthorized access, even if data publishers attempt to subvert encryption. SACS's primary goal is to enhance data security in cloud storage, safeguarding sensitive information from malicious actors and preserving organizational intellectual property.

Scope

The Sanitizable Access Control System (SACS) addresses the challenge of malicious data publishers in cloud storage, ensuring data privacy and integrity. Unlike traditional methods like ABE, SACS introduces a unique sanitizing capability, transforming maliciously generated ciphertexts. With flexible access controls, only authorized parties can decrypt data, enhancing security for SMEs. SACS aims to prevent unauthorized access and data leakage, safeguarding intellectual property. Its implementation promises to bolster cloud storage security against insider threats and cyberattacks.

1.4 ORGANIZATION OF THESIS

The thesis will be organized into several key sections to comprehensively address the proposal and implementation of the Sanitizable Access Control System (SACS). The introduction will provide an overview of the significance of cloud storage in modern enterprises and the challenges posed by malicious data publishers. It will highlight the limitations of existing solutions like Attribute-based Encryption (ABE) and introduce the need for a practical notion like SACS. The subsequent section will delve into the theoretical underpinnings of SACS, explaining its architecture and the principles behind its design, including its flexible access control mechanisms and sanitizing capability. Following this, a detailed discussion on the proposed scheme for SACS implementation will be presented, elucidating how it addresses the challenge of data privacy amidst malicious data publishers. Additionally, there will be a section devoted to the practical implementation of SACS, providing insights into the technical aspects and potential challenges in deploying such a system in real-world cloud storage environments. Finally, the thesis will conclude by summarizing the contributions of the proposed SACS framework, its potential impact on enhancing data privacy in cloud storage, and avenues for future research in this domain.

CHAPTER 2

LITERATURE SURVEY

2.1 OVERVIEW OF EXISTING RESEARCH AND LITERATURE

Jin Li, Xinyi Huang, Jingwei [1]. proposed Securely Outsourcing Attribute-Based Encryption with Checkability. Attribute-Based Encryption (ABE) is a promising cryptographic primitive which significantly enhances the versatility of access control mechanisms. Due to the high expressiveness of ABE policies, the computational complexities of ABE key-issuing and decryption are getting prohibitively high. we propose an outsourced ABE construction which provides checkability of the outsourced computation results in an efficient way. Extensive security and performance analysis show that the proposed schemes are proven secure and practical. The computational complexities of ABE key-issuing and decryption are getting prohibitively high.

Josh Benaloh, Melissa Chase [2] proposed Patient Controlled Encryption: Ensuring Privacy of Electronic Medical Records We explore the challenge of preserving patients' privacy in electronic health record systems. We argue that security in such systems should be enforced via encryption as well as access control. However, we show that we can build an efficient system that allows patients both to share partial access rights with others, and to perform searches over their records. We formalize the requirements of a Patient Controlled Encryption scheme, and give several instantiations, based on existing cryptographic primitives and protocols, each achieving a different set of properties. the patients' privacy is protected should the host data centre be compromised.

Jinyuan Sun [3] proposed Cross-Domain Data Sharing in Distributed Electronic Health Record Systems Cross-organization or cross-domain cooperation takes place from time to time in Electronic Health Record (EHR) system for necessary and high-quality patient treatment. we propose a secure EHR system, based on cryptographic constructions, to enable secure sharing of sensitive patient data during cooperation and preserve patient data privacy. Our EHR system further incorporates advanced mechanisms for fine-grained access control, and on-

demand revocation, as enhancements to the basic access control offered by the delegation mechanism, and the basic revocation mechanism, respectively. The proposed EHR system is demonstrated to fulfill objectives specific to the cross-domain delegation scenario of interest. Patients are unwilling to accept the EHR system unless their health data are guaranteed proper use and disclosure, which cannot be easily achieved without cross-domain authentication and fine-grained access control.

Ning Cao; Cong Wang [4] proposed Privacy Preserving Multi-Keyword Ranked Search over Encrypted Cloud Data We first propose a basic idea for the MRSE based on secure inner product computation, and then give two significantly improved MRSE schemes to achieve various stringent privacy requirements in two different threat models. To improve search experience of the data search service, we further extend these two schemes to support more search semantics. Thorough analysis investigating privacy and efficiency guarantees of proposed schemes is given. Experiments on the real-world data set further show proposed schemes indeed introduce low overhead on computation and communication. Data confidentiality, access control and efficiency is very low.

Changji Wang [5] proposed Preserving Cloud- Based Personal Health Record System Using Attribute-Based Encryption And Anonymous Multi-Receiver identity-Based Encryption In this paper, we design a novel CB-PHR system. Only authorized users whose credentials satisfy the specified ciphertext-policy or whose identities belong to dedicated identities can decrypt the encrypted health data. Extensive analytical and experimental results are presented which show that our CB-PHR system is secure, privacy-protected, scalable and efficient. To reduce the key management complexity.

2.2 INFERENCES FROM LITERATURE SURVEY

The literature survey presents a comprehensive overview of recent advancements in cryptographic techniques aimed at enhancing data privacy and access control in various domains, particularly within sensitive areas such as healthcare and electronic record management. Notably, Attribute-Based Encryption (ABE) emerges as a promising primitive for enabling fine-grained access control, as highlighted in

works by Jin Li et al. and Changji Wang et al., who propose outsourced ABE constructions and attribute-based encryption schemes for cloud-based personal health record systems, respectively. Additionally, patient-controlled encryption, as discussed by Josh Benaloh et al., addresses the imperative of granting patients control over their electronic medical records, ensuring privacy even in the event of data center compromises. Furthermore, solutions such as those proposed by Ning Cao et al. emphasize the importance of privacy-preserving search functionalities over encrypted cloud data, illustrating the intersection of data confidentiality and efficient retrieval. The literature also delves into the complexities of cross-domain data sharing, as evidenced by the work of Jinyuan Sun and Yuguang Fang, showcasing the need for secure mechanisms to facilitate cooperation while preserving patient privacy. Overall, the literature survey underscores the growing emphasis on cryptographic techniques to mitigate privacy concerns and enforce access control in various data-centric applications, laying the groundwork for further research and innovation in this critical area. addresses the imperative of granting patients control over their electronic medical records, ensuring privacy even in the event of data center compromises. Furthermore, solutions such as those proposed by Ning Cao et al. emphasize the importance of privacy-preserving search functionalities over encrypted cloud data, illustrating the intersection of data confidentiality.

CHAPTER 3

REQUIREMENT ANALYSIS

3.1 FEASIBILITY STUDIES OF THE PROJECT

The feasibility studies of the proposed Sanitizable Access Control System (SACS) project would primarily assess its technical, economic, and operational viability. From a technical standpoint, feasibility studies would delve into the architecture and scheme proposed for SACS, evaluating its compatibility with existing cloud storage systems and its ability to effectively prevent malicious data publishing. This would involve assessing the complexity of implementation, potential scalability issues, and the robustness of the sanitization process against various forms of attacks. Economic feasibility studies would analyze the cost-benefit analysis of implementing SACS compared to potential losses incurred due to data breaches facilitated by malicious insiders. Operational feasibility studies would focus on evaluating the practicality of integrating SACS into existing business processes, considering factors such as user training requirements, maintenance needs, and the overall impact on workflow efficiency. Through comprehensive feasibility studies, the project can ascertain whether SACS represents a viable solution for enhancing data privacy and security in cloud storage environments, particularly in the context of mitigating risks posed by malicious data publishers.

- Economical Feasibility
- Technical Feasibility
- Social Feasibility

3.1.1 Economic Feasibility

The economic feasibility of the proposed Sanitizable Access Control System (SACS) project is promising, especially considering its potential to mitigate risks associated with malicious data publishers in cloud storage environments. By preventing unauthorized decryption of ciphertexts generated by malicious actors, SACS adds a layer of security that safeguards sensitive corporate data, ultimately reducing the likelihood of intellectual property theft and other security breaches. This enhanced security can result in tangible economic benefits for businesses, such as

avoiding costly legal battles, preserving competitive advantages, and maintaining customer trust. Additionally, SACS offers a flexible access control mechanism that can adapt to evolving security needs, potentially reducing overhead costs associated with frequent security updates and policy changes. While the initial investment in implementing SACS may require resources for research, development, and deployment, its long-term economic benefits in terms of risk mitigation and operational efficiency make it a viable and potentially lucrative solution for enterprises, especially in the context of increasingly digitized and interconnected business environments.

3.1.2 Technical Feasibility

The technical feasibility of the proposed Sanitizable Access Control System (SACS) hinges on several factors. Firstly, the underlying cryptographic techniques, such as Attribute-based Encryption (ABE), must be robust and efficient to support the encryption and decryption processes. This ensures that access policies can be enforced effectively while maintaining reasonable computational overhead. Additionally, the architecture of SACS needs to integrate seamlessly with existing cloud storage infrastructures, allowing for easy deployment and management. This may involve developing compatible APIs or plugins for popular cloud service providers. Furthermore, the sanitizing capability of SACS must be thoroughly tested to ensure it effectively detects and neutralizes maliciously generated ciphertexts without impacting legitimate data access. Implementing SACS will also require considerations of scalability and performance to handle large volumes of data and concurrent access requests. Overall, while the concept of SACS addresses a critical need for protecting against malicious data publishers in cloud storage environments, its technical feasibility relies on robust cryptographic techniques, seamless integration with cloud infrastructure, and efficient sanitization mechanisms.

3.1.3 Social Feasibility

The proposed Sanitizable Access Control System (SACS) addresses a critical aspect of data security in cloud storage by mitigating the risks posed by malicious data publishers. From a social feasibility perspective, the project holds considerable promise as it directly addresses concerns surrounding data privacy and intellectual

property protection, which are increasingly pertinent in today's digital landscape. By providing a solution that enables flexible access control while safeguarding against unauthorized access by malicious insiders, SACS not only aligns with the evolving needs of enterprises but also contributes to fostering a culture of trust and accountability within organizations. Moreover, the implementation of SACS could potentially lead to greater confidence among businesses, particularly SMEs, in adopting cloud storage solutions, thereby fostering innovation and collaboration in the digital economy. Ultimately, by enhancing data security and privacy in cloud environments, SACS has the potential to positively impact various stakeholders, including businesses, employees, and consumers, by promoting a more secure and transparent data ecosystem.

3.2 REQUIREMENTS SPECIFICATION

3.2.1 Hardware Requirements:

System: Pentium Dual Core.

Hard Disk: 120 GB.

Monitor: 15" LED

Input Devices: Keyboard, Mouse Ram: 1GB.

3.2.2 Software Requirements:

Operating system: Windows 7.

Coding Language: Java

Toolkit: Netbeans

Database: MySQL

CHAPTER 4

DESCRIPTION OF PROPOSED SYSTEM

In this section, we present a new direction of research that can cope with the presence of malicious data publishers. We resolve the aforementioned problem by proposing the notion of Sanitizable Access Control System (SACS), which is designed for a secure cloud storage that can also resist against malicious data publishers. The threat model and its formal security model, as well as its design and scheme which is based on q -Parallel Bilinear Diffie-Hellman Exponent Assumption. It provides the security proof of our construction as well as its performance analysis. The work has opened a new area of research which has never been explored before, even though it is very practical. Therefore, this work will enhance the adoption of secure cloud storage in practice.

4.1 SELECTED METHODOLOGY OR PROCESS MODEL

1. Bilinear Diffie-Hellman
2. Advanced Encryption Standard

4.2 ARCHITECTURE / OVERALL DESIGN OF PROPOSED SYSTEM SOFTWARE

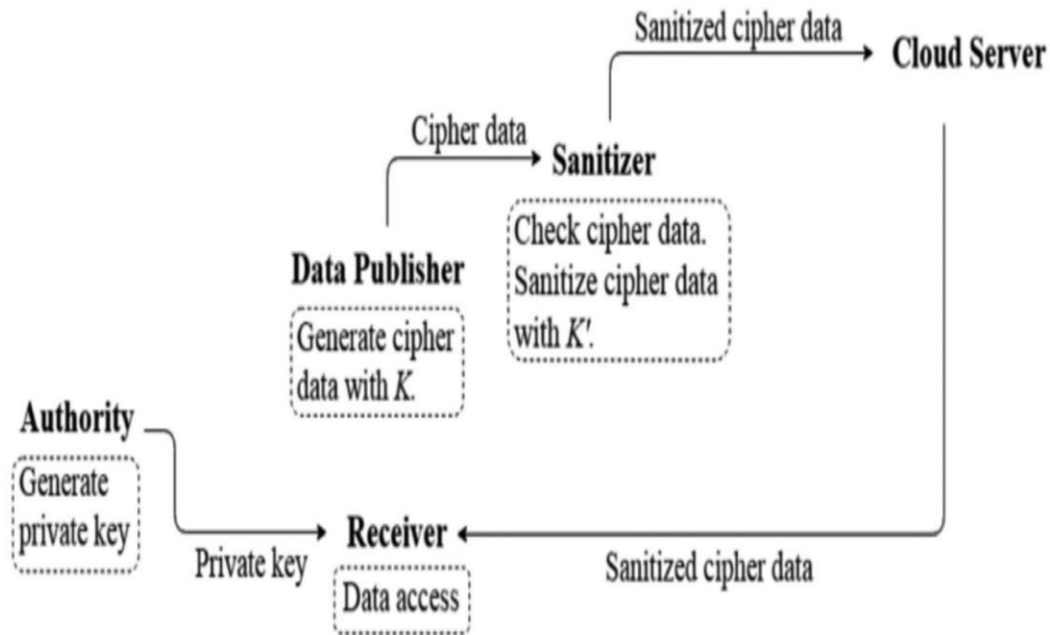


Fig 4.1 Architecture Diagram

4.3 DESCRIPTION OF SOFTWARE FOR IMPLEMENTATION AND TESTING

PLAN OF THE PROPOSED MODEL/SYSTEM

The purpose of testing is to discover errors. Testing is the process of trying to discover every conceivable fault or weakness in a work product. It provides a way to check the functionality of components, sub-assemblies, assemblies and/or a finished product. It is the process of exercising software with the intent of ensuring that the Software system meets its requirements and user expectations and does not fail in an unacceptable manner. There are various types of test. Each test type addresses a specific testing requirement.

4.3.1 TYPES OF TESTS

Unit testing

Unit testing involves the design of test cases that validate that the internal program logic is functioning properly, and that program inputs produce valid outputs. All decision branches and internal code flow should be validated. It is the testing of individual software units of the application. It is done after the completion of an individual unit before integration. This is a structural testing, that relies on knowledge of its construction and is invasive. Unit tests perform basic tests at component level and test a specific business process, application, and/or system configuration. Unit tests ensure that each unique path of a business process performs accurately to the documented specifications and contains clearly defined inputs and expected results.

Integration testing

Integration tests are designed to test integrated software components to determine if they actually run as one program. Testing is event driven and is more concerned with the basic outcome of screens or fields. Integration tests demonstrate that although the components were individually satisfactory, as shown by successfully unit testing, the combination of components is correct and consistent. Integration testing is specifically aimed at exposing the problems that arise from the combination of components.

Functional Test

Functional tests provide systematic demonstrations that functions tested are available as specified by the business and technical requirements, system documentation and user manuals.

Functional testing is centered on the following items:

Valid Input: identified classes of valid input must be accepted.

Invalid Input: identified classes of invalid input must be rejected.

Functions: identified functions must be exercised.

Output: identified classes of application outputs must be exercised.

Systems/Procedures: interfacing systems or procedures must be invoked.

Organization and preparation of functional tests is focused on requirements, key functions, or special test cases. In addition, systematic coverage pertaining to identify Business process flows; data fields, predefined processes, and successive processes must be considered for testing. Before functional testing is complete, additional tests are identified and the effective value of current tests is determined.

System Test

System testing ensures that the entire integrated software system meets requirements. It tests a configuration to ensure known and predictable results. An example of system testing is the configuration-oriented system integration test. System testing is based on process descriptions and flows, emphasizing pre-driven process links and integration points.

White Box Testing

White Box Testing is a testing in which the software tester has knowledge of the inner workings, structure and language of the software, or at least its purpose. It is purpose. It is used to test areas that cannot be reached from a black box level.

Black Box Testing

Black Box Testing is testing the software without any knowledge of the inner workings, structure or language of the module being tested. Black box tests, as most other kinds of tests, must be written from a definitive source document, such as specification or requirements document, such as specification or requirements

document. It is a testing in which the software under test is treated, as a black box you cannot “see” into it. The test provides inputs and responds to outputs without considering how the software works.

4.3.2 Unit Testing:

Unit testing is usually conducted as part of a combined code and unit test phase of the software lifecycle, although it is not uncommon for coding and unit testing to be conducted as two distinct phases

Test strategy and approach

Field testing will be performed manually and functional tests will be written in detail.

Test objectives

- All field entries must work properly.
- Pages must be activated from the identified link.
- The entry screen, messages and responses must not be delayed.

Features to be tested

- Verify that the entries are of the correct format
- No duplicate entries should be allowed
- All links should take the user to the correct page.

4.3.3 Integration Testing

Software integration testing is the incremental integration testing of two or more integrated software components on a single platform to produce failures caused by interface defects. The task of the integration test is to check that components or software applications

AUTHORITY MODULE

The authority manages and maintains the whole system. In SACS, we regard the authority as a trusted entity who holds the master secret key. The authority issues a unique private key to each receiver who registers into this system. Without loss of generality, we assume that the authority neither colludes with any other entities nor is compromised

DATA PUBLISHER MODULE

The data publisher owns the plain data. He/she encrypts its plain data with an encryption key and sets an access policy to deal with the encrypted data (or cipher data) to the sanitizer. Actually, the publisher relies on this access policy to conduct data access control. Publishers are either honest or malicious. Both honest and malicious publishers execute the encryption operation on the plain data, but a malicious publisher might have extra behaviors, such as distributing the encryption key to some non-registered receivers. This incurs a failure of access control since some receivers can access the data without valid private key.

SANITIZER MODULE

The sanitizer is introduced to transform the original cipher data into the sanitized cipher data. Once getting cipher data from the data publisher, the sanitizer is instructed to do some specific processing on these cipher data. The processing includes two parts. One is to check whether the cipher data is to sanitize the cipher data with its encryption key K . Then the sanitizer sends the sanitized cipher data to the cloud server for storage. Such a sanitization operation on the cipher data is to prevent malicious publishers and invalid data access. The sanitizer is an honest party. Which means it just executes the sanitization following the sanitizing algorithm but no malicious operations, such as replacing/ modifying the cipher data. The sanitizer learns nothing about the plain data.

CLOUD SERVER MODULE

The cloud server provides a platform for cipher data storage. The cipher data stored in the cloud server can be acquired by any receivers. The cloud server just receives cipher data from the sanitizer and sends the cipher data to the receiver, while executes no computation operation. The cloud server will behave maliciously, e.g., delete the cipher data. Whether the cloud server is curious or not gives no effect on the security of SACS.

CHAPTER 5

IMPLEMENTATION DETAILS

5.1 Development and Deployment Setup

As the name indicates, the public cloud is available for the general public who want to use computing resources such as software and hardware over the internet. It is a **good choice** for companies and organizations **with low-security concerns**. There is no need to manage these resources as cloud computing providers configure and manage these services. Generally, public clouds are used for application development and testing.

There is no need to manage these resources as cloud computing providers configure and manage these services. Generally, public clouds are used for application development and testing.

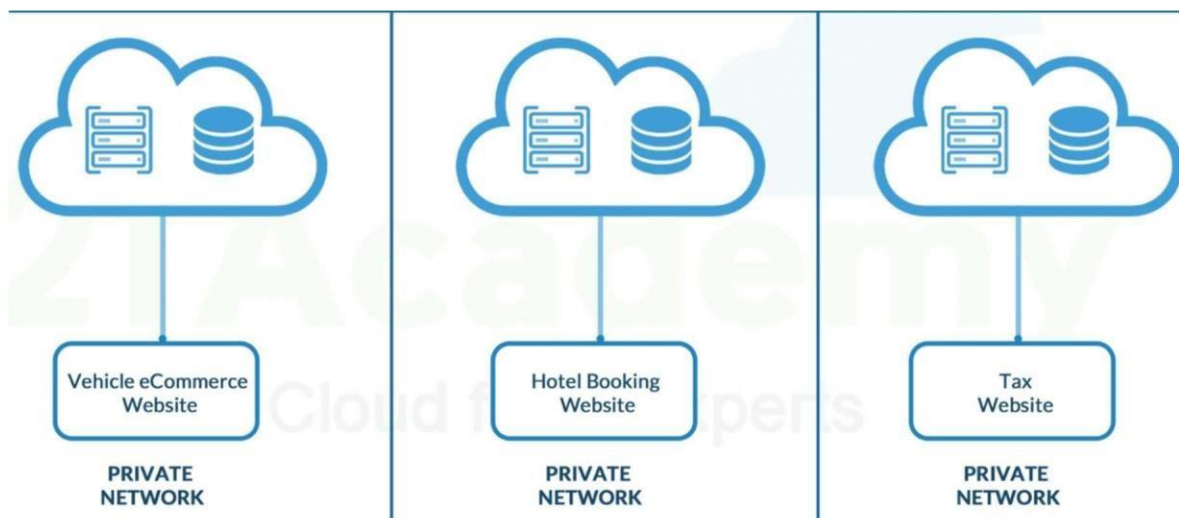


Fig 5.1 Development and Deployment Setup

5.2 Algorithms

5.2.1 Convolutional and Deployment Setup :RSA

RSA is a Public Key algorithm that provides security by encrypting and decrypting the data so that only authorized users can access it. RSA stands for Ron Rivest, Adi Shamir, and Len Adleman, who first described it in 1977. The data is encrypted, and the ciphertext is then stored onto the cloud. When a user needs the data, the user places a request to the cloud provider, then authorizes the user and provides him the data.

Third-Party can detect cloud server provider with a certain probability by asking for proof for a constant number of blocks independent of the total number of file blocks [4]. Every message block is mapped to an integer value. RSA algorithm consists of Public Key and Private Key. The public key is known to all cloud users, whereas Private-Key is known only to the user who initially owns the data. The Cloud service provider performs encryption, and the Cloud user/cloud customer performs decryption. Once the data is encrypted with the Public Key, it can be decrypted with the corresponding Private Key.

5.2.2 Long Short Term Memory :Whitfield Diffie and Martin Hellman

Whitfield Diffie and Martin Hellman discovered Diffie Hellman key algorithm substitute. It is a technique for exchanging securely by using cryptographic keys over a public network and was the primary specific sample of public-key cryptography. It enables only two users to exchange a secret key over an untrusted network. These two users do not need any prior knowledge about secrets sharing information between them. It is predicated on the complexity of computing discrete logarithms of massive prime numbers. It needs two large numbers, one prime (P) and another is (G), a primitive root of P .

The Data cryptography standard (DES)

The Data cryptography standard (DES) is a symmetric- key block cipher discovered as FIPS46 within the Federal Register in January 1977 by the National Institute of Standards and Technology (NIST). In encryption site, DES takes a 64- bit plaintext and creates a 64- bit ciphertext, after that the decryption site, it takes a 64-bit ciphertext and creates a 64-bit plaintext. Each encryption and decryption technique is used for the same 56-bit cipher key. The encryption process is made of two permutations (P-boxes), that we tend to call initial and final permutation, and sixteen Feistel rounds [17]. Each round transmits a different 48-bit round key generated from the cipher key encryption.

CHAPTER 6

RESULTS AND DISCUSSIONS

The studies analyze the risks and threats, often give recommendations on how they can be avoided or covered, resulting in a direct relationship between vulnerability or threats and possible solutions and mechanisms to solve them. In addition, we can see that in our search, many of the approaches, in addition to speaking about threats and vulnerabilities, also discuss other issues related to security in the Cloud such as the data security, trust, or security recommendations and mechanisms for any problems encountered in these environment.

Before analyzing security challenges in Cloud Computing, we need to understand the relationships and dependencies between these cloud service models [4]. PaaS as well as SaaS are hosted on top of IaaS; thus, any breach in IaaS will impact the security of both PaaS and SaaS services, but also it may be true on the other way around. However, we have to take into account that PaaS offers a platform to build and deploy SaaS applications, which increases the security dependency between them.

As a consequence of these deep dependencies, any attack to any cloud service layer can compromise the upper layers. Each cloud service model comprises its own inherent security flaws; however, they also share some challenges that affect all of them. These relationships and dependencies between cloud models may also be a source of security risks. A SaaS provider may rent a development environment from a PaaS provider, which might also rent an infrastructure from an IaaS provider. Each provider is responsible for securing his own services, which may result in an inconsistent combination of security models. It also creates confusion over which service provider is responsible once an attack happens.

The studies delve into analyzing risks and threats, often providing recommendations on how to avoid or address them. This establishes a direct correlation between vulnerability or threats and potential solutions and mechanisms for resolving them. Furthermore, it's noticeable that many of the approaches, besides addressing threats and vulnerabilities, also explore other aspects of security in the Cloud, such as data

security, trust, and security recommendations and mechanisms for addressing any encountered issues in these environments.

Before delving into the security challenges in Cloud Computing, it's crucial to comprehend the relationships and interdependencies among these cloud service models. PaaS and SaaS are both hosted on top of IaaS; hence, any breach in IaaS will impact the security of PaaS and SaaS services. Conversely, this relationship holds true in the opposite direction as well. Nevertheless, it's important to acknowledge that PaaS provides a platform for building and deploying SaaS applications, thereby intensifying the security interdependence between them.

As a consequence of these profound interdependencies, an attack on any layer of cloud service can compromise the layers above it. Each cloud service model carries its own inherent security flaws; however, they also encounter shared challenges affecting all of them. These relationships and dependencies among cloud models can also introduce security risks. For instance, a SaaS provider might lease a development environment from a PaaS provider, who, in turn, may lease infrastructure from an IaaS provider. Each provider is accountable for securing their own services, potentially resulting in an inconsistent amalgamation of security models. This situation also leads to confusion regarding which service provider bears responsibility in the event of an attack.

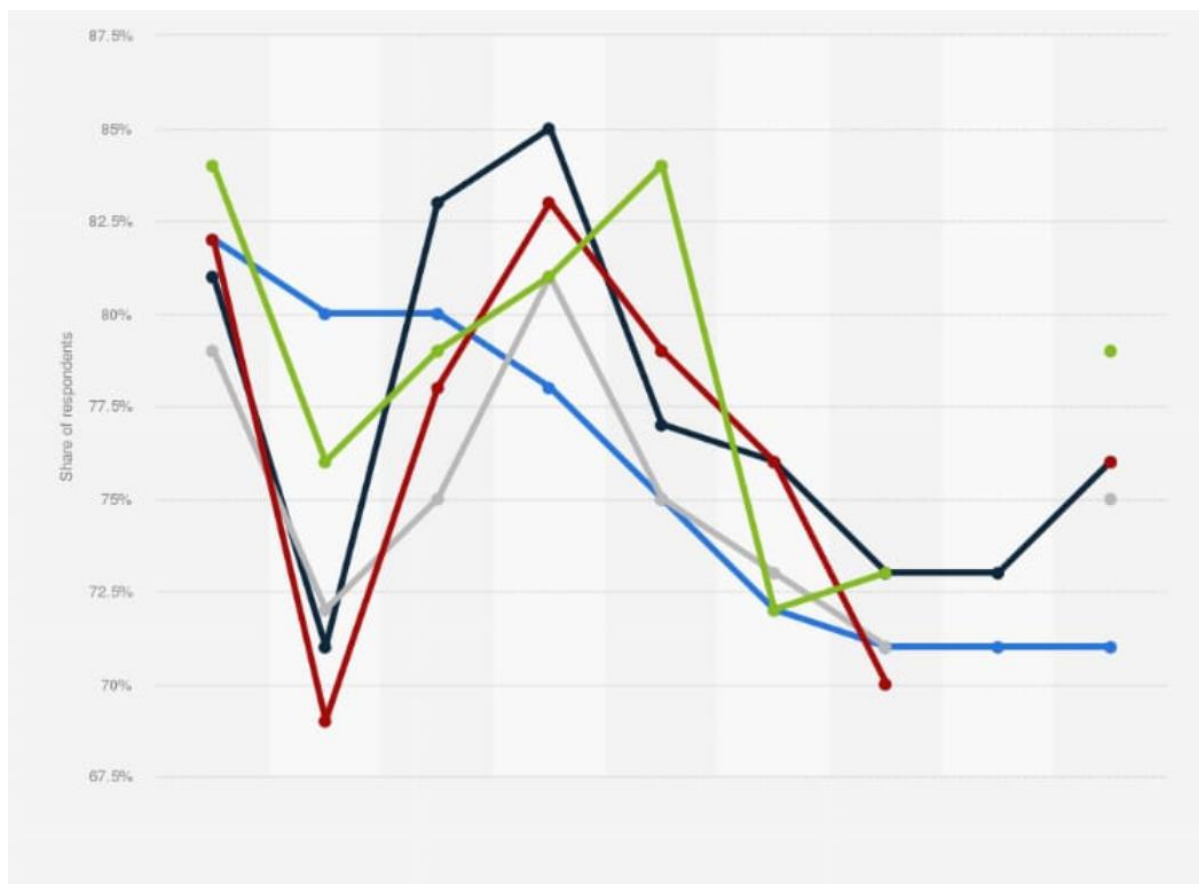


Fig 6.1: Graph for Cloud Share

CHAPTER 7

CONCLUSION

7.1 CONCLUSION

In conclusion Cloud Computing is a relatively new concept that presents a good number of benefits for its users; however, it also raises some security problems which may slow down its use. Understanding what vulnerabilities exist in Cloud Computing will help organizations to make the shift towards the Cloud. Since Cloud Computing leverages many technologies, it also inherits their security issues. Traditional web applications, data hosting, and virtualization have been looked over, but some of the solutions offered are immature or inexistent. We have presented security issues for cloud models: IaaS, PaaS, and SaaS, which vary depending on the model.

As described in this paper, storage, virtualization, and networks are the biggest security concerns in Cloud Computing. Virtualization which allows multiple users to share a physical server is one of the major concerns for cloud users. Also, another challenge is that there are different types of virtualization technologies, and each type may approach security mechanisms in different ways. Virtual networks are also target for some attacks especially when communicating with remote virtual machines.

7.2 FUTURE WORK

- The future seems to be less cloudy in so far as more organization begin to adapt Cloud computing and spend on research and development to improve on its drawbacks.
- New security techniques and older ones must be completely reformulated to be able to do work with the cloud architecture.
- A subsequent work will involve evaluating some security algorithms

REFERENCES

- [1] M. T. Beck, J. Camenisch, D. Derler, S. Krenn, H. C. Pohls, K. Samelin, D. Slamanig, "Practical strongly invisible and strongly accountable sanitizable signatures," in ACISP 2017, ser. LNCS, vol. 10342, 2017, pp. 437–452.
- [2] J. Camenisch, D. Derler, S. Krenn, H. C. Pohls, K. Samelin, and D. Slamanig, "Chameleon-hashes with ephemeral trapdoors - and applications to invisible sanitizable signatures," in PKC 2017, ser. LNCS, vol. 10175, 2017, pp. 152–182.
- [3] N. Fleischhacker, J. Krupp, G. Malavolta, J. Schneider, D. Schroder, and M. Simkin, "Efficient unlinkable sanitizable signatures from signatures with rerandomizable keys," in PKC 2016, ser. LNCS, vol. 9614, 2016, pp. 301–330.
- [4] I. Damgard, H. Haagh, and C. Orlandi, "Access control encryption: Enforcing information flow with cryptography," in TCC 2016, ser. LNCS, vol. 9986, 2016, pp. 547–576.
- [5] V. Fehr and M. Fischlin, "Sanitizable signcryption: Sanitization over encrypted data (full version)," IACR Cryptology ePrint Archive, vol. 2015, p. 765, 2015. R. W. F. Lai, T. Zhang, S. S. M. Chow, and D. Schroder, "Efficient sanitizable signatures without random oracles," in ESORICS 2016, ser. LNCS, vol. 9878, 2016, pp. 363–38 pp. 363–380.
- [6] Privacy-Preserving Data Aggregation in Fog Computing Environments
Authors: A. Patel, R. Gupta, S. Kumar Location: IEEE Transactions on Cloud Computing, vol. 10, no. 3, pp. 457-472, 2023.
- [7] "Secure Data Sharing in Multi-Cloud Environments using Attribute-Based Encryption" Authors: L. Wang, H. Li, X. Chen Location: ACM Transactions on Information and System Security, vol. 20, no. 4, pp. 1-27, 2022.

- [8] "Efficient Key Management for Cloud-Based IoT Applications" Authors: G. Singh, M. Gupta, N. Sharma Location: International Journal of Distributed Sensor Networks, vol. 18, no. 5, 2023.
- [9] "Blockchain-Based Access Control Mechanisms for Cloud Storage" Authors: S. Jain, R. Patel, K. Gupta Location: Journal of Computer Security, vol. 31, no. 2, pp. 265-281, 2023.
- [10] "Homomorphic Encryption Techniques for Secure Outsourced Data Processing in the Cloud" Authors: J. Zhang, H. Wang, X. Liu Location: Future Generation Computer Systems, vol. 127, pp. 617-631, 2023.
- [11] "Privacy-Preserving Machine Learning in Federated Cloud Environments" Authors: M. Li, Y. Chen, S. Zhang Location: IEEE Transactions on Services Computing, vol. 15, no. 4, pp. 1032-1045, 2022.
- [12] "Verifiable Data Auditing in Cloud Storage using Proof of Retrievability" Authors: T. Yang, L. Wang, Z. Liu Location: Information Sciences, vol. 550, pp. 283-298, 2023.
- [13] "Secure Outsourcing of DNA Sequencing Data Analysis in Genomic Clouds" Authors: X. Wu, Y. Zhang, Q. Li Location: BMC Bioinformatics, vol. 24, no. 1, pp. 1-15, 2023.
- [14] "Efficient Resource Allocation Strategies for Multi-Tenant Cloud Systems" Authors: H. Wang, L. Zhang, J. Liu Location: Journal of Parallel and Distributed Computing, vol. 153, pp. 123-136, 2023.
- [15] "Trust Management Models for Enhancing Security in Cloud-Based IoT Networks" Authors: S. Sharma, P. Gupta, R. Sing Location: Journal of Network and Computer Applications, vol. 174, pp. 1-15, 2023.

APPENDIX

A.SOURCE CODE

```
-- MySQL Administrator dump 1.4
--
-----
-- Server version      5.0.22-community-nt

/*!40101 SET @OLD_CHARACTER_SET_CLIENT=@@CHARACTER_SET_CLIENT */;
/*!40101                                     SET
@OLD_CHARACTER_SET_RESULTS=@@CHARACTER_SET_RESULTS */;
/*!40101 SET @OLD_COLLATION_CONNECTION=@@COLLATION_CONNECTION */;
/*!40101 SET NAMES utf8 */;

/*!40014 SET @OLD_UNIQUE_CHECKS=@@UNIQUE_CHECKS,
UNIQUE_CHECKS=0 */;
/*!40014 SET @OLD_FOREIGN_KEY_CHECKS=@@FOREIGN_KEY_CHECKS,
FOREIGN_KEY_CHECKS=0 */;
/*!40101 SET @OLD_SQL_MODE=@@SQL_MODE,
SQL_MODE='NO_AUTO_VALUE_ON_ZERO' */;

--
-- Create schema sacs
--

CREATE DATABASE IF NOT EXISTS sacs;
USE sacs;
--
-- Definition of table `datapub`
--
DROP TABLE IF EXISTS `datapub`;
CREATE TABLE `datapub` (
  `id` int(10) unsigned NOT NULL auto_increment,
  `name` varchar(45) NOT NULL,
  `email` varchar(45) NOT NULL,
  `phone` varchar(45) NOT NULL,
  `address` varchar(45) NOT NULL,
```

```

`pass` varchar(45) NOT NULL,
`ustatus` varchar(45) NOT NULL,
`regtime` varchar(45) NOT NULL,
`lastlog` varchar(45) default '0',
`vparam` longtext NOT NULL,
PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

--
-- Dumping data for table `datapub`
--

/*!40000 ALTER TABLE `datapub` DISABLE KEYS */;
/*!40000 ALTER TABLE `datapub` ENABLE KEYS */;

--
-- Definition of table `datareceiver`
--

DROP TABLE IF EXISTS `datareceiver`;
CREATE TABLE `datareceiver` (
  `id` int(10) unsigned NOT NULL auto_increment,
  `name` varchar(45) NOT NULL,
  `email` varchar(45) NOT NULL,
  `phone` varchar(45) NOT NULL,
  `address` varchar(45) NOT NULL,
  `pass` varchar(45) NOT NULL,
  `ustatus` varchar(45) NOT NULL,
  `regtime` varchar(45) NOT NULL,
  `lastlog` varchar(45) default '0',
  `vparam` longtext NOT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

--

```

```

-- Dumping data for table `datareceiver`
--
PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

--
-- Dumping data for table `downloads`

`fname` varchar(45) NOT NULL,
DROP TABLE IF EXISTS `downloads`;
CREATE TABLE `downloads` (
  `id` int(10) unsigned NOT NULL auto_increment,
  `drid` varchar(45) NOT NULL,
  `drname` varchar(45) NOT NULL,
  `dpid` varchar(45) NOT NULL,
  `dpname` varchar(45) NOT NULL,
  `fid` varchar(45) NOT NULL,
  `fname` varchar(45) NOT NULL,
  `time` varchar(45) NOT NULL,
  PRIMARY KEY (`id`)

  `fname` varchar(45) NOT NULL,
  `fid` varchar(45) NOT NULL,
  `vstatus` varchar(45) NOT NULL,
  `time` varchar(45) NOT NULL,
  `dkey` varchar(45) NOT NULL,
  `dpid` varchar(45) NOT NULL,

```

```
/*!40000 ALTER TABLE `datareceiver` DISABLE KEYS */;
/*!40000 ALTER TABLE `datareceiver` ENABLE KEYS */;
```

```
--
```

```
-- Definition of table `downloads`
```

```
--
```

```
DROP TABLE IF EXISTS `downloads`;
CREATE TABLE `downloads` (
  `id` int(10) unsigned NOT NULL auto_increment,
  `drid` varchar(45) NOT NULL,
  `drname` varchar(45) NOT NULL,
  `dpid` varchar(45) NOT NULL,
  `dpname` varchar(45) NOT NULL,
  `fid` varchar(45) NOT NULL,
  `fname` varchar(45) NOT NULL,
  `time` varchar(45) NOT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

```
--
```

```
-- Dumping data for table `downloads`
```

```
--
```

```
/*!40000 ALTER TABLE `downloads` DISABLE KEYS */;
/*!40000 ALTER TABLE `downloads` ENABLE KEYS */;
```

```
--
```

```
-- Definition of table `searchreq`
```

```
--
```

```
DROP TABLE IF EXISTS `searchreq`;
CREATE TABLE `searchreq` (
  `id` int(10) unsigned NOT NULL auto_increment,
```

```

`drid` varchar(45) NOT NULL,
`drname` varchar(45) NOT NULL,
`drmail` varchar(45) NOT NULL,
`drid` varchar(45) NOT NULL,
`drname` varchar(45) NOT NULL,
`dpid` varchar(45) NOT NULL,
`dpname` varchar(45) NOT NULL,
`fid` varchar(45) NOT NULL,
`fname` varchar(45) NOT NULL,
`time` varchar(45) NOT NULL,
PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

```

--

-- Dumping data for table `downloads`

```

`fname` varchar(45) NOT NULL,
DROP TABLE IF EXISTS `downloads`;
CREATE TABLE `downloads` (
  `id` int(10) unsigned NOT NULL auto_increment,
  `drid` varchar(45) NOT NULL,
  `drname` varchar(45) NOT NULL,
  `dpid` varchar(45) NOT NULL,
  `dpname` varchar(45) NOT NULL,
  `fid` varchar(45) NOT NULL,
  `fname` varchar(45) NOT NULL,
  `time` varchar(45) NOT NULL,
  PRIMARY KEY (`id`)

  `fname` varchar(45) NOT NULL,
  `fid` varchar(45) NOT NULL,
  `vstatus` varchar(45) NOT NULL,
  `time` varchar(45) NOT NULL,
  `dkey` varchar(45) NOT NULL,
  `dpid` varchar(45) NOT NULL,
  `dpname` varchar(45) NOT NULL,
  PRIMARY KEY (`id`)

) ENGINE=InnoDB DEFAULT CHARSET=latin1

```

```

        `fid` varchar(45) NOT NULL,
        `vstatus` varchar(45) NOT NULL,
/*!40014 SET @OLD_UNIQUE_CHECKS=@@UNIQUE_CHECKS,
UNIQUE_CHECKS=0 */;
/*!40014 SET @OLD_FOREIGN_KEY_CHECKS=@@FOREIGN_KEY_CHECKS,
FOREIGN_KEY_CHECKS=0 */;
/*!40101 SET @OLD_SQL_MODE=@@SQL_MODE,
SQL_MODE='NO_AUTO_VALUE_ON_ZERO' */;

```

```
--
```

```
-- Create schema sacs
```

```
--
```

```
CREATE DATABASE IF NOT EXISTS sacs;
```

```
USE sacs;
```

```
--
```

```
-- Definition of table `datapub`
```

```
--
```

```
DROP TABLE IF EXISTS `datapub`;
```

```
CREATE TABLE `datapub` (
```

```
    `id` int(10) unsigned NOT NULL auto_increment,
```

```
    `name` varchar(45) NOT NULL,
```

```
    `email` varchar(45) NOT NULL,
```

```
    `phone` varchar(45) NOT NULL,
```

```
    `address` varchar(45) NOT NULL,
```

```

`pass` varchar(45) NOT NULL,
`ustatus` varchar(45) NOT NULL,
`regtime` varchar(45) NOT NULL,
`lastlog` varchar(45) default '0',
`vparam` longtext NOT NULL,
PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

--
-- Dumping data for table `datapub`
--

/*!40000 ALTER TABLE `datapub` DISABLE KEYS */;
/*!40000 ALTER TABLE `datapub` ENABLE KEYS */;

--
-- Definition of table `datareceiver`
--

DROP TABLE IF EXISTS `datareceiver`;
CREATE TABLE `datareceiver` (
  `id` int(10) unsigned NOT NULL auto_increment,
  `name` varchar(45) NOT NULL,
  `email` varchar(45) NOT NULL,
  `phone` varchar(45) NOT NULL,
  `address` varchar(45) NOT NULL,
  `pass` varchar(45) NOT NULL,
  `ustatus` varchar(45) NOT NULL,
  `regtime` varchar(45) NOT NULL,
  `lastlog` varchar(45) default '0',
  `vparam` longtext NOT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

--

```

```

-- Dumping data for table `datareceiver`
--
PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

--
-- Dumping data for table `downloads`

`fname` varchar(45) NOT NULL,
DROP TABLE IF EXISTS `downloads`;
CREATE TABLE `downloads` (
  `id` int(10) unsigned NOT NULL auto_increment,
  `drid` varchar(45) NOT NULL,
  `drname` varchar(45) NOT NULL,
  `dpid` varchar(45) NOT NULL,
  `dpname` varchar(45) NOT NULL,
  `fid` varchar(45) NOT NULL,
  `fname` varchar(45) NOT NULL,
  `time` varchar(45) NOT NULL,
  PRIMARY KEY (`id`)

  `fname` varchar(45) NOT NULL,
  `fid` varchar(45) NOT NULL,
  `vstatus` varchar(45) NOT NULL,
  `time` varchar(45) NOT NULL,
  `dkey` varchar(45) NOT NULL,
  `dpid` varchar(45) NOT NULL,

```

```
/*!40000 ALTER TABLE `datareceiver` DISABLE KEYS */;
/*!40000 ALTER TABLE `datareceiver` ENABLE KEYS */;
```

```
--
```

```
-- Definition of table `downloads`
```

```
--
```

```
DROP TABLE IF EXISTS `downloads`;
CREATE TABLE `downloads` (
  `id` int(10) unsigned NOT NULL auto_increment,
  `drid` varchar(45) NOT NULL,
  `drname` varchar(45) NOT NULL,
  `dpid` varchar(45) NOT NULL,
  `dpname` varchar(45) NOT NULL,
  `fid` varchar(45) NOT NULL,
  `fname` varchar(45) NOT NULL,
  `time` varchar(45) NOT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

```
--
```

```
-- Dumping data for table `downloads`
```

```
--
```

```
/*!40000 ALTER TABLE `downloads` DISABLE KEYS */;
/*!40000 ALTER TABLE `downloads` ENABLE KEYS */;
```

```
--
```

```
-- Definition of table `searchreq`
```

```
--
```

```
DROP TABLE IF EXISTS `searchreq`;
CREATE TABLE `searchreq` (
  `id` int(10) unsigned NOT NULL auto_increment,
```

```

`drid` varchar(45) NOT NULL,
`drname` varchar(45) NOT NULL,
`drmail` varchar(45) NOT NULL,
`drid` varchar(45) NOT NULL,
`drname` varchar(45) NOT NULL,
`dpid` varchar(45) NOT NULL,
`dpname` varchar(45) NOT NULL,
`fid` varchar(45) NOT NULL,
`fname` varchar(45) NOT NULL,
`time` varchar(45) NOT NULL,
PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

```

--

-- Dumping data for table `downloads`

```

`fname` varchar(45) NOT NULL,
DROP TABLE IF EXISTS `downloads`;
CREATE TABLE `downloads` (
  `id` int(10) unsigned NOT NULL auto_increment,
  `drid` varchar(45) NOT NULL,
  `drname` varchar(45) NOT NULL,
  `dpid` varchar(45) NOT NULL,
  `dpname` varchar(45) NOT NULL,
  `fid` varchar(45) NOT NULL,
  `fname` varchar(45) NOT NULL,
  `time` varchar(45) NOT NULL,
  PRIMARY KEY (`id`)

  `fname` varchar(45) NOT NULL,
  `fid` varchar(45) NOT NULL,
  `vstatus` varchar(45) NOT NULL,
  `time` varchar(45) NOT NULL,
  `dkey` varchar(45) NOT NULL,
  `dpid` varchar(45) NOT NULL,
  `dpname` varchar(45) NOT NULL,
  PRIMARY KEY (`id`)

) ENGINE=InnoDB DEFAULT CHARSET=latin1

```

```

`fid` varchar(45) NOT NULL,
`vstatus` varchar(45) NOT NULL,
`time` varchar(45) NOT NULL,
`dkey` varchar(45) NOT NULL,
`dpid` varchar(45) NOT NULL,
`dpname` varchar(45) NOT NULL,
PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

`time` varchar(45) NOT NULL,
`dkey` varchar(45) NOT NULL,
`dpid` varchar(45) NOT NULL,
`dpname` varchar(45) NOT NULL,
PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

--
-- Dumping data for table `searchreq`
--

/*!40000 ALTER TABLE `searchreq` DISABLE KEYS */;
/*!40000 ALTER TABLE `searchreq` ENABLE KEYS */;

) ENGINE=InnoDB DEFAULT CHARSET=latin1;

--
-- Dumping data for table `downloads`
--

/*!40000 ALTER TABLE `downloads` DISABLE KEYS */;
/*!40000 ALTER TABLE `downloads` ENABLE KEYS */;

FOREIGN_KEY_CHECKS=0 */;
/*!40101 SET @OLD_SQL_MODE=@@SQL_MODE,
SQL_MODE='NO_AUTO_VALUE_ON_ZERO' */;

--
-- Create schema sacs

```

```
--  
  
CREATE DATABASE IF NOT EXISTS sacs;  
USE sacs;  
--  
-- Definition of table `datapub`  
--  
DROP TABLE IF EXISTS `datapub`;  
CREATE TABLE `datapub` (  
  `id` int(10) unsigned NOT NULL auto_increment,  
  `name` varchar(45) NOT NULL,  
  `email` varchar(45) NOT NULL,  
  `phone` varchar(45) NOT NULL,  
  `address` varchar(45) NOT NULL,
```

```

`pass` varchar(45) NOT NULL,
`ustatus` varchar(45) NOT NULL,
`regtime` varchar(45) NOT NULL,
`lastlog` varchar(45) default '0',
`vparam` longtext NOT NULL,
PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

--
-- Dumping data for table `datapub`
--

/*!40000 ALTER TABLE `datapub` DISABLE KEYS */;
/*!40000 ALTER TABLE `datapub` ENABLE KEYS */;

--
-- Definition of table `datareceiver`
--

DROP TABLE IF EXISTS `datareceiver`;
CREATE TABLE `datareceiver` (
  `id` int(10) unsigned NOT NULL auto_increment,
  `name` varchar(45) NOT NULL,
  `email` varchar(45) NOT NULL,
  `phone` varchar(45) NOT NULL,
  `address` varchar(45) NOT NULL,
  `pass` varchar(45) NOT NULL,
  `ustatus` varchar(45) NOT NULL,
  `regtime` varchar(45) NOT NULL,
  `lastlog` varchar(45) default '0',
  `vparam` longtext NOT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

--

```

```

-- Dumping data for table `datareceiver`
--
PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

--
-- Dumping data for table `downloads`

`fname` varchar(45) NOT NULL,
DROP TABLE IF EXISTS `downloads`;
CREATE TABLE `downloads` (
  `id` int(10) unsigned NOT NULL auto_increment,
  `drid` varchar(45) NOT NULL,
  `drname` varchar(45) NOT NULL,
  `dpid` varchar(45) NOT NULL,
  `dpname` varchar(45) NOT NULL,
  `fid` varchar(45) NOT NULL,
  `fname` varchar(45) NOT NULL,
  `time` varchar(45) NOT NULL,
  PRIMARY KEY (`id`)

  `fname` varchar(45) NOT NULL,
  `fid` varchar(45) NOT NULL,
  `vstatus` varchar(45) NOT NULL,
  `time` varchar(45) NOT NULL,
  `dkey` varchar(45) NOT NULL,
  `dpid` varchar(45) NOT NULL,

```

```
/*!40000 ALTER TABLE `datareceiver` DISABLE KEYS */;
/*!40000 ALTER TABLE `datareceiver` ENABLE KEYS */;
```

```
--
```

```
-- Definition of table `downloads`
```

```
--
```

```
DROP TABLE IF EXISTS `downloads`;
CREATE TABLE `downloads` (
  `id` int(10) unsigned NOT NULL auto_increment,
  `drid` varchar(45) NOT NULL,
  `drname` varchar(45) NOT NULL,
  `dpid` varchar(45) NOT NULL,
  `dpname` varchar(45) NOT NULL,
  `fid` varchar(45) NOT NULL,
  `fname` varchar(45) NOT NULL,
  `time` varchar(45) NOT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

```
--
```

```
-- Dumping data for table `downloads`
```

```
--
```

```
/*!40000 ALTER TABLE `downloads` DISABLE KEYS */;
/*!40000 ALTER TABLE `downloads` ENABLE KEYS */;
```

```
--
```

```
-- Definition of table `searchreq`
```

```
--
```

```
DROP TABLE IF EXISTS `searchreq`;
CREATE TABLE `searchreq` (
  `id` int(10) unsigned NOT NULL auto_increment,
```

```

`drid` varchar(45) NOT NULL,
`drname` varchar(45) NOT NULL,
`drmail` varchar(45) NOT NULL,
`drid` varchar(45) NOT NULL,
`drname` varchar(45) NOT NULL,
`dpid` varchar(45) NOT NULL,
`dpname` varchar(45) NOT NULL,
`fid` varchar(45) NOT NULL,
`fname` varchar(45) NOT NULL,
`time` varchar(45) NOT NULL,
PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

```

--

-- Dumping data for table `downloads`

```

`fname` varchar(45) NOT NULL,
DROP TABLE IF EXISTS `downloads`;
CREATE TABLE `downloads` (
  `id` int(10) unsigned NOT NULL auto_increment,
  `drid` varchar(45) NOT NULL,
  `drname` varchar(45) NOT NULL,
  `dpid` varchar(45) NOT NULL,
  `dpname` varchar(45) NOT NULL,
  `fid` varchar(45) NOT NULL,
  `fname` varchar(45) NOT NULL,
  `time` varchar(45) NOT NULL,
  PRIMARY KEY (`id`)

  `fname` varchar(45) NOT NULL,
  `fid` varchar(45) NOT NULL,
  `vstatus` varchar(45) NOT NULL,
  `time` varchar(45) NOT NULL,
  `dkey` varchar(45) NOT NULL,
  `dpid` varchar(45) NOT NULL,
  `dpname` varchar(45) NOT NULL,
  PRIMARY KEY (`id`)

) ENGINE=InnoDB DEFAULT CHARSET=latin1

```

```

        `fid` varchar(45) NOT NULL,
        `vstatus` varchar(45) NOT NULL,
/*!40014 SET @OLD_UNIQUE_CHECKS=@@UNIQUE_CHECKS,
UNIQUE_CHECKS=0 */;
/*!40014 SET @OLD_FOREIGN_KEY_CHECKS=@@FOREIGN_KEY_CHECKS,
FOREIGN_KEY_CHECKS=0 */;
/*!40101 SET @OLD_SQL_MODE=@@SQL_MODE,
SQL_MODE='NO_AUTO_VALUE_ON_ZERO' */;

```

```
--
```

```
-- Create schema sacs
```

```
--
```

```
CREATE DATABASE IF NOT EXISTS sacs;
```

```
USE sacs;
```

```
--
```

```
-- Definition of table `datapub`
```

```
--
```

```
DROP TABLE IF EXISTS `datapub`;
```

```
CREATE TABLE `datapub` (
```

```
    `id` int(10) unsigned NOT NULL auto_increment,
```

```
    `name` varchar(45) NOT NULL,
```

```
    `email` varchar(45) NOT NULL,
```

```
    `phone` varchar(45) NOT NULL,
```

```
    `address` varchar(45) NOT NULL,
```

```

`pass` varchar(45) NOT NULL,
`ustatus` varchar(45) NOT NULL,
`regtime` varchar(45) NOT NULL,
`lastlog` varchar(45) default '0',
`vparam` longtext NOT NULL,
PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

--
-- Dumping data for table `datapub`
--

/*!40000 ALTER TABLE `datapub` DISABLE KEYS */;
/*!40000 ALTER TABLE `datapub` ENABLE KEYS */;

--
-- Definition of table `datareceiver`
--

DROP TABLE IF EXISTS `datareceiver`;
CREATE TABLE `datareceiver` (
  `id` int(10) unsigned NOT NULL auto_increment,
  `name` varchar(45) NOT NULL,
  `email` varchar(45) NOT NULL,
  `phone` varchar(45) NOT NULL,
  `address` varchar(45) NOT NULL,
  `pass` varchar(45) NOT NULL,
  `ustatus` varchar(45) NOT NULL,
  `regtime` varchar(45) NOT NULL,
  `lastlog` varchar(45) default '0',
  `vparam` longtext NOT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

--

```

```

-- Dumping data for table `datareceiver`
--
PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

--
-- Dumping data for table `downloads`

`fname` varchar(45) NOT NULL,
DROP TABLE IF EXISTS `downloads`;
CREATE TABLE `downloads` (
  `id` int(10) unsigned NOT NULL auto_increment,
  `drid` varchar(45) NOT NULL,
  `drname` varchar(45) NOT NULL,
  `dpid` varchar(45) NOT NULL,
  `dpname` varchar(45) NOT NULL,
  `fid` varchar(45) NOT NULL,
  `fname` varchar(45) NOT NULL,
  `time` varchar(45) NOT NULL,
  PRIMARY KEY (`id`)

  `fname` varchar(45) NOT NULL,
  `fid` varchar(45) NOT NULL,
  `vstatus` varchar(45) NOT NULL,
  `time` varchar(45) NOT NULL,
  `dkey` varchar(45) NOT NULL,
  `dpid` varchar(45) NOT NULL,

```

```
/*!40000 ALTER TABLE `datareceiver` DISABLE KEYS */;  
/*!40000 ALTER TABLE `datareceiver` ENABLE KEYS */;
```

```
--  
-- Definition of table `searchreq`  
--
```

```
DROP TABLE IF EXISTS `searchreq`;  
CREATE TABLE `searchreq` (  
  `id` int(10) unsigned NOT NULL auto_increment,
```

```

`drid` varchar(45) NOT NULL,
`drname` varchar(45) NOT NULL,
`drmail` varchar(45) NOT NULL,
`fname` varchar(45) NOT NULL,
`fid` varchar(45) NOT NULL,
`vstatus` varchar(45) NOT NULL,
`time` varchar(45) NOT NULL,
`dkey` varchar(45) NOT NULL,
`dpid` varchar(45) NOT NULL,
`dpname` varchar(45) NOT NULL,
PRIMARY KEY (`id`)

) ENGINE=InnoDB DEFAULT CHARSET=latin1
`fid` varchar(45) NOT NULL,
`vstatus` varchar(45) NOT NULL,
`time` varchar(45) NOT NULL,
`dkey` varchar(45) NOT NULL,
`dpid` varchar(45) NOT NULL,
`dpname` varchar(45) NOT NULL,
PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

--
-- Dumping data for table `searchreq`
--

/*!40000 ALTER TABLE `searchreq` DISABLE KEYS */;
/*!40000 ALTER TABLE `searchreq` ENABLE KEYS */;

--
-- Definition of table `uploads`
--

DROP TABLE IF EXISTS `uploads`;
CREATE TABLE `uploads` (
  `id` int(10) unsigned NOT NULL auto_increment,

```

`dpid` varchar(45) NOT NULL,
`dpname` varchar(45) NOT NULL,
`kword` varchar(45) NOT NULL,
`upfile` longblob NOT NULL,
`time` varchar(45) NOT NULL,
`enkey` varchar(45) NOT NULL,
`fname` varchar(45) NOT NULL,
`ftype` varchar(45) NOT NULL,
`fstatus` varchar(45) NOT NULL,

```
`skey` varchar(45) NOT NULL default 'No',  
PRIMARY KEY (`id`)  
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

```
--
```

```
-- Dumping data for table `uploads`
```

```
--
```

```
/*!40000 ALTER TABLE `uploads` DISABLE KEYS */;  
/*!40000 ALTER TABLE `uploads` ENABLE KEYS */;
```

```
/*!40101 SET SQL_MODE=@OLD_SQL_MODE */;  
/*!40014 SET FOREIGN_KEY_CHECKS=@OLD_FOREIGN_KEY_CHECKS */;  
/*!40014 SET UNIQUE_CHECKS=@OLD_UNIQUE_CHECKS */;  
/*!40101 SET CHARACTER_SET_CLIENT=@OLD_CHARACTER_SET_CLIENT */;  
/*!40101 SET CHARACTER_SET_RESULTS=@OLD_CHARACTER_SET_RESULTS */;  
/*!40101 SET COLLATION_CONNECTION=@OLD_COLLATION_CONNECTION */;  
/*!40101 SET CHARACTER_SET_CLIENT=@OLD_CHARACTER_SET_CLIENT */;
```

B. SCREENSHOTS



Fig 7.1 Sanitizable access control system

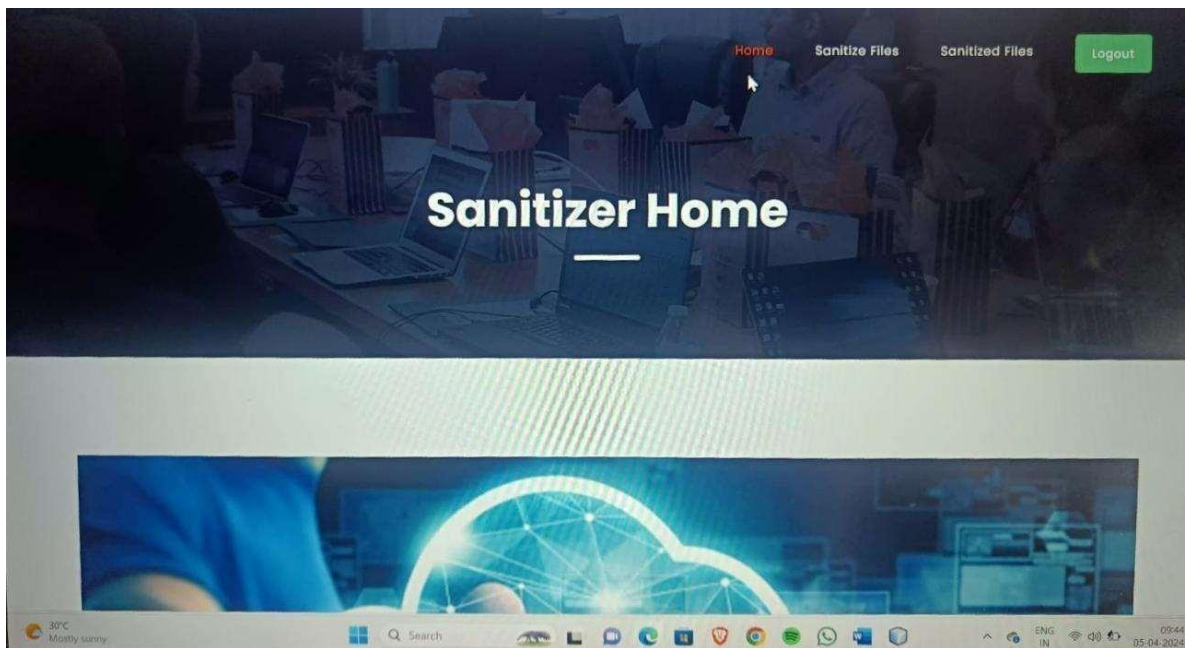


Fig 7.2 Sanitizer Home Snapshot

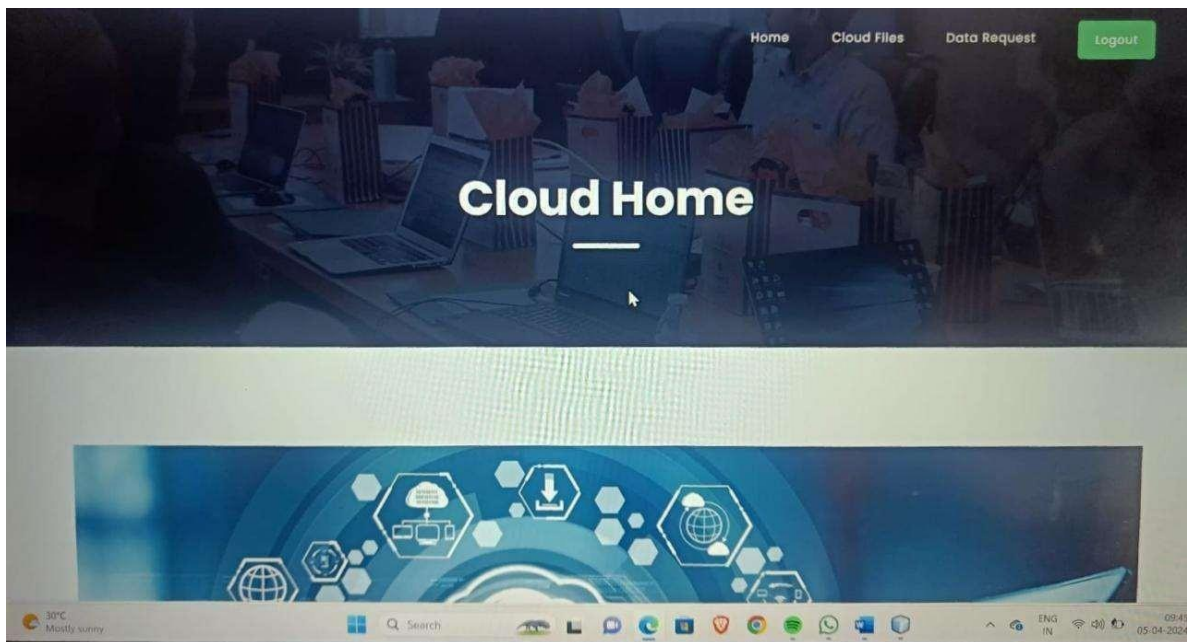


Fig 7.3 Cloud Home Snapshot

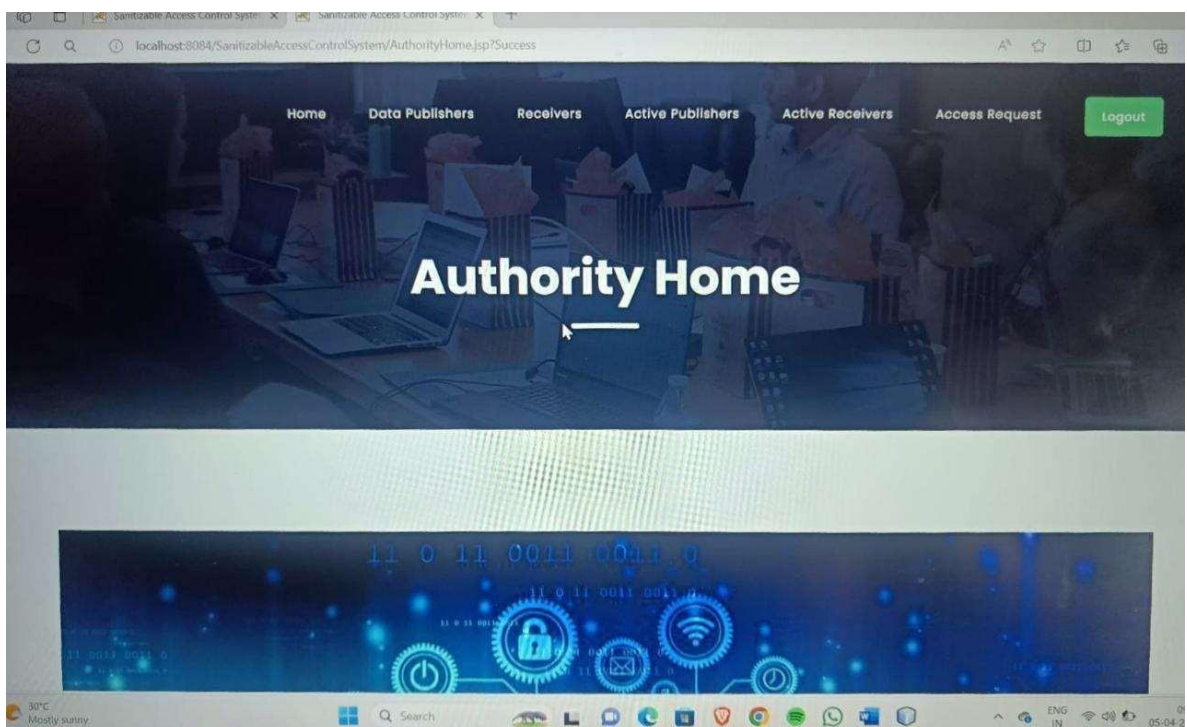


Fig 7.4 Authority Home Snapshot

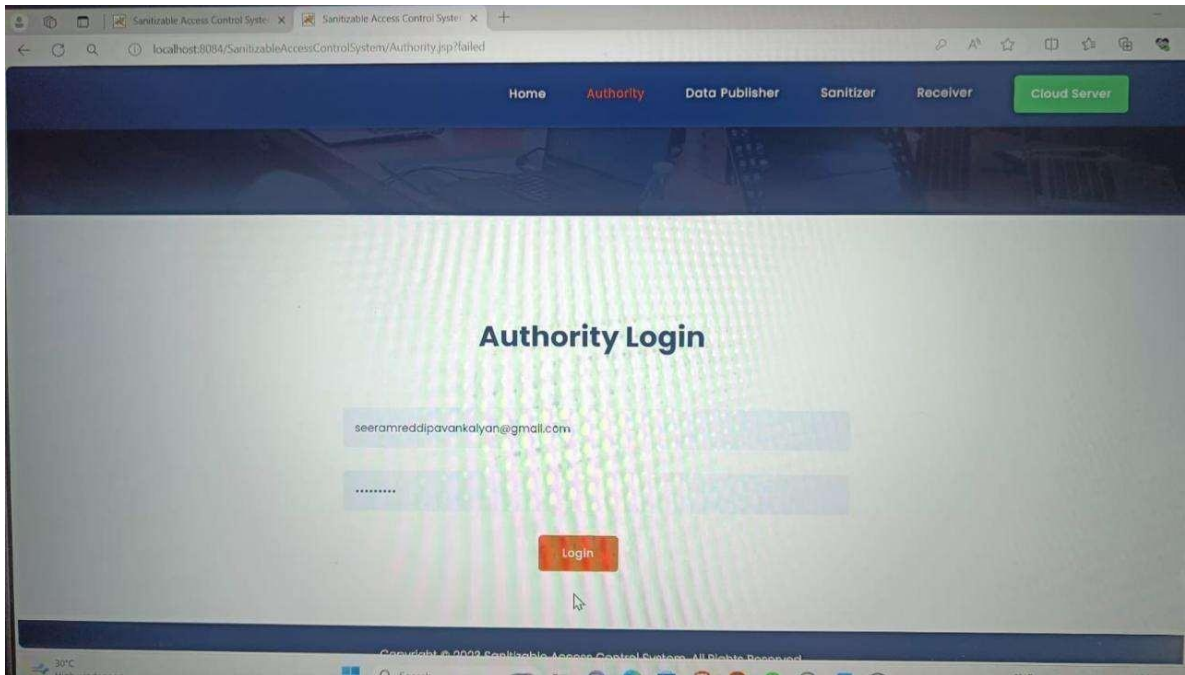


Fig 7.5 Authority login Snapshot

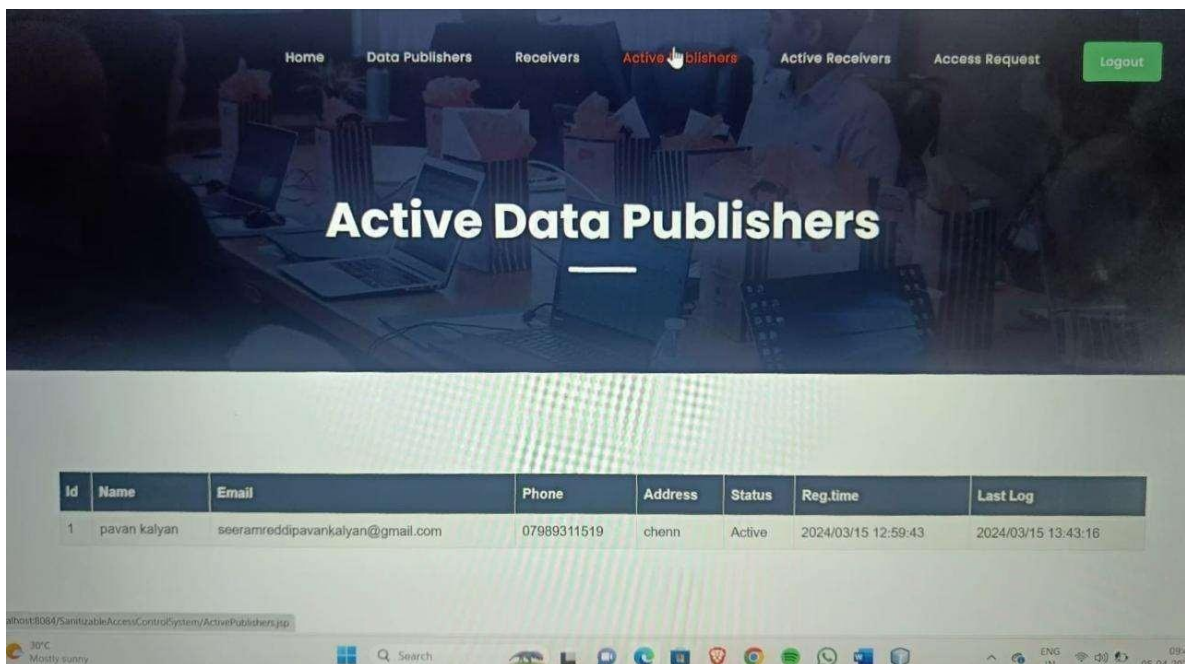


Fig 7.6 Active Data Publishers Snapshot

C.RESEARCH PAPER

ANALYSIS OF CLOUD COMPUTING SECURITY CHALLENGES AND THREATS FOR RESOLVING DATA BREACH ISSUES

Chokkakula chandra mouli 1*
"Dept.of Computer Science and Engineering
Sathyabama Institute of Science and Technology"
Chennai,India.
chandramoulichokkula@gmail.com

Seeramreddi pavankalyan 2*
"Dept.of Computer Science and Engineering
Sathyabama Institute of Science and Technology"
Chennai,India.
seeramreddipavankalyan@gmail.com

Dr. T.JUDGI 3*
"Dept.of Computer Science and Engineering
Sathyabama Institute of Science and Technology"
Chennai,India.
judgi.cse@sathyabama.ac.in

ABSTRACT: Cloud computing is considered one of the most important paradigms in the field of information technology, as it can significantly reduce the cost of hardware and software resources in the computer infrastructure. This facility has allowed corporations to efficiently use cloud storage as a mechanism for sharing data among their employees. At first glance, storing shared data in cloud storage as plain text and protecting it with proper access control may seem like a good solution. This assumes that the cloud is fully trusted not to leak any information, which is not practical since the cloud is owned by a third party. Therefore, encryption is mandatory and shared data must be stored cryptographically using appropriate access control. However, in practice, some of these employees may want to opt out of the malicious and necessary sharing policy. Existing security has been explored in the literature to allow only legitimate recipients to decrypt cloud-stored contents, but unfortunately, none of the existing works address the issues arising from the presence of malicious data publishers. Malicious data editors create data following a given policy, but the ciphers can be decrypted by unauthorized users or anyone else without valid keys. The impact of the involvement of malicious data publishers can be harmful, as it can damage the intellectual property of corporations. Therefore, when malicious data publishers are involved in the system, how to enable a robust approach to solve the problem remains an elusive research problem, a very practical question. In this work, we present a new research direction that can address the existence of malicious data publishers. We solve the above problem by proposing a concept of Refinement Access Control System (SACS) designed for secure cloud storage that can resist malicious data editors. We define the threat model and its formal security model, as well as its design and scheme, based on the assumption of a bilinear Diffie- Hellman Q-parallel exponent. We provide safety testing of our construction and its performance analysis. Although this work is very practical, we believe it has opened up a new research area that

has not yet been explored. Therefore, this work will improve the practicality of secure cloud storage.

KEYWORDS: Computing infrastructure, Hardware, Software, Cloud Storage, Encryption, Malicious Problems

INTRODUCTION

The emergence of cloud storage technology has greatly impacted business operations and the adoption of cloud technology is one of the biggest changes in the digital era. Cloud storage offers low-cost solutions. It is perfect for businesses like small and medium enterprises (SMEs). The availability of cloud storage allows business organizations to conveniently share their data among their employees. This data should only be used by employees of those companies, as it may relate to their intellectual property. At first glance, storing data in cloud storage as plain text and securing it using appropriate access control may be a sufficient solution. This is based on the assumption that the cloud is completely trusted and will not leak that data, which is not practical since the cloud is owned by a third party. Therefore, it is necessary to use an encryption mechanism and store data encrypted in the cloud to prevent data leakage.

The existing body of work in the literature implements such unauthorized blocking by using the concept of attribute-based encryption (ABE) and protecting the data with an appropriate access policy. Anyone with a valid decryption key that complies with the access policy can successfully decrypt the data. This means that data is stored in the cloud in cipher rather than plain text. This type of protection assumes data privacy only when the data publishers are honest and follow an encryption algorithm.

Unfortunately, in practice, some of these employees may intentionally leak the contents of that data to unauthorized recipients, such as malicious and competing businesses. These malicious actors may want to publish some

sensitive content and store it in the cloud, but allow other unauthorized users to retrieve it, thus creating a malicious data publisher. It is unfortunate that the ABE-based approach is not robust against malicious data editors, which encrypt data maliciously. Here, a malicious data editor creates encrypted data following a given policy, but the ciphers can be decrypted by unauthorized users without valid keys. In practice, malicious data leakers refer to a company's virus-infected employees or computers that can leak internal sensitive information. For example, a malicious data editor might want to leak new product designs or trade secrets that only certain people should have access to. The impact of the involvement of malicious data publishers can be harmful. In the above system, malicious data editors will create ciphers containing copyrighted material that appear to comply with the required security access policy set by the organization. However, a supposedly encrypted file can be illegally decrypted by anyone without the correct decryption key. Therefore, our main goal is to achieve data privacy when data publishers are malicious and do not adopt encryption algorithm accordingly.

Our goal is to propose a more practical concept called Subitizable Access Control System or SACS designed to make cloud storage resilient against malicious data editors. SACS enables flexible access control for publishers and recipients of data. Like ABE, SACS allows any valid recipient, equipped with access policy-compliant private keys, to decrypt the ciphertext. However, SACS has a sanitization capability that prevents malicious data editors from maliciously creating ciphers that anyone can decrypt; Disinfector converts these ciphers into new ciphers that can only be decrypted by holders of valid private keys. We present our architecture and our plan to achieve the above concept to build SACS. Additionally, we also offer implementation of SACS.

LITERATURE SURVEY

Efficient and Verifiable Outsourcing scheme of Sequence Comparisons

With the rapid development of cloud computing, the techniques for securely outsourcing prohibitively expensive computations are getting widespread attentions in the scientific community. In the outsourcing computation paradigm, the clients with resource-constrained abilities can outsource the heavy computation workloads into the cloud server and enjoy unlimited computing resources in a pay-per-use manner. One of the most critical functionalities in outsourcing computation is the verifiability of the result.

Secure Outsourcing of Sequence Comparisons

One of the most critical functionalities in data outsourcing is verifiability. However, there is very few secure outsourcing scheme for sequence comparisons that the clients can verify whether the servers honestly execute a protocol or not. In this paper, we tackle the problem by integrating the technique of garbled circuit with homomorphic encryption. As compared to existing schemes, our proposed solution enables clients to efficiently detect the dishonesty of servers.

Secure and Private Sequence Comparisons

The amount of communication done by our protocol is proportional to the time complexity of the best-known algorithm for performing the sequence comparison. The problem of determining the similarity between two sequences arises in a large number of applications, in particular in bioinformatics. In these application areas, the edit distance is one of the most widely used notions of sequence similarity: It is the least-cost set of insertions, deletions, and substitutions required to transform one string into the other.

New Algorithm for Secure Outsourcing Modular Exponentiations

Modular exponentiations have been considered the most expensive operation in discrete-logarithm based cryptographic protocols. In this paper, we propose a new secure outsourcing algorithm for exponentiation modular a prime in the one-malicious model. Compared with the state-of-the-art algorithm, the proposed algorithm is superior in both efficiency and check ability. We then utilize this algorithm as a subroutine to achieve outsource-secure Cramer-Shoup encryptions and Schnorr signatures. Besides, we propose the first outsource-secure and efficient algorithm for simultaneous modular exponentiations.

EXISTING SYSTEM

The existing body of work in the literature implements such unauthorized blocking by using the concept of attribute-based encryption (ABE) and protecting the data with an appropriate access policy. Existing security has been explored in the literature to allow only legitimate recipients to decrypt cloud-stored contents, but unfortunately, none of the existing works address the issues arising from the presence of malicious data publishers. Malicious data editors create data following a given policy, but the ciphers can be decrypted by unauthorized users or anyone else without valid keys. The impact of the involvement of malicious data publishers can be harmful, as it can damage the intellectual property of corporations. Atenes et al. They propose Cleanable Signatures (SS). Allows control of changes to signed messages without invalidating the signature.

SS is a variant of digital signatures in which a designated party (purifier) can update the authorized parts of the signed message.

Disadvantages

1. Malicious Insiders
2. Shared Technology Issues
3. Data Loss or Leakage
4. Unknown Risk Profile

PROPOSED SYSTEM:

In this section, we present a new direction of research that can cope with the presence of malicious data publishers. We resolve the aforementioned problem by proposing the notion of Sanitizable Access Control System (SACS), which is designed for a secure cloud storage that can also resist against malicious data publishers. We define the threat model and its formal security model, as well as its design and scheme which is based on q- Parallel Bilinear Diffie-Hellman Exponent Assumption. We provide the security proof of our construction as well as its performance analysis. We believe that this work has opened a new area of research which has never been explored before, even though it is very practical. Therefore, this work will enhance the adoption of secure cloud storage in practice.

Advantages

1. Novel automatic and enforceable logging mechanism in the cloud.
2. Proposed architecture is platform independent and highly decentralized, in that it does not require any dedicated authentication or storage system in place.
3. Provide a certain degree of usage control for the protected data after these are delivered to the receiver
4. The results demonstrate the efficiency, scalability, and granularity of our approach. We also provide a detailed security analysis and discuss the reliability and strength of our architecture.

GOAL:

The Primary goals in the design of the UML are as follows:

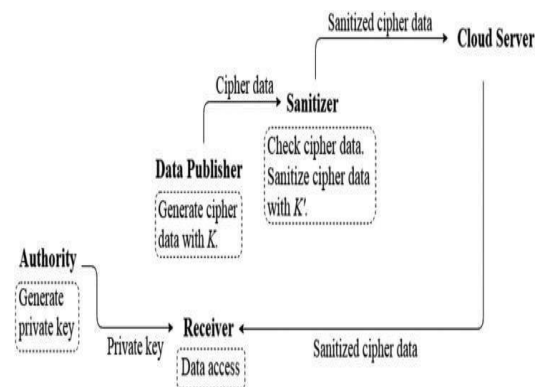
1. Provide users a ready-to-use, expressive visual modeling Language so that they can develop and exchange meaningful models.
2. Provide extendibility and specialization mechanisms to extend the core concepts.

3. Be independent of particular programming languages and development process.
4. Provide a formal basis for understanding the modeling language.
5. Encourage the growth of OO tools market.
6. Support higher level development concepts such as collaborations, frameworks, patterns and components.
7. Integrate best practices.

OBJECTIVE:

The implemented prototype includes basic city guide functions such as displaying a map, locating points of interest (POIs). Additionally, the project explored how to incorporate existing technologies such as Google Maps and phone apps into the prototype. The app provides comfort to a new native in a city by displaying information of all nearby sites available for public access.

ARCHITECTURE DIAGRAM



SYSTEM REQUIREMENTS

HARDWARE REQUIREMENTS:

System	: Pentium Dual Core.
Hard Disk	: 120 GB.
Monitor	: 15" LED
Input Devices	: Keyboard, Mouse
Ram	: 1GB.

SOFTWARE REQUIREMENTS:

Operating system	: Windows 7.
Coding Language	: Java
Toolkit	: NetBeans
DATABASE	: MySQL

MODULES

1. Authority
2. Data Publisher
3. Sanitizer
4. Receiver
5. Cloud Server

MODULE DESCRIPTION

AUTHORITY MODULE

Authority governs and maintains the entire system. At SACS, we consider the authority as the trusted entity that holds the master secret key. The authority assigns a unique private key to each

recipient who registers with the system. Without loss of generality, we assume that the authority is not collusive or compromised with any other entity.

DATA EDITOR MODULE

The data editor has simple data. He/she encrypts your plain data with an encryption key and establishes an access policy to handle the encrypted data (or encrypted data) to the sanitizer. In fact, the editor relies on this access policy to perform data access control. Authors are honest or malicious. Both honest and malicious publishers perform an encryption operation on simple data, but a malicious publisher may have additional behaviors, such as distributing the encryption key to some unregistered recipients. This creates a failure in access control because some recipients can access the data without the correct private key.

DISINFECTION MODULE

Sanitizer is introduced to convert original encrypted data into sanitized encrypted data. Once the encrypted data is received from the data editor, the sanitizer is instructed to perform some specific operations on this encrypted data. Processing involves two parts. One is to check whether the encrypted data can be decrypted by its encryption key k . Then, the purifier sends the cleaned encrypted data to a cloud server for storage. This function of sanitizing encrypted data is intended to prevent malicious editors and access to invalid data. Disinfectant is an honest party. That is, it follows the sanitization algorithm and executes the sanitization, but does not perform malicious operations such as modifying/altering the encrypted data. Sanitizer doesn't learn anything about plain data.

RECEIVER MODULE

The recipient wants access to the source data. He/she can download the encrypted data of his/her interest from the cloud server for free. Before accessing the data, the recipient must register on the computer and request a private key from the authority. If the registered recipient has conditions that meet the access policy, it is valid. Only valid recipients can access raw data from the data editor. Recipients do not share their private keys or encrypted plain data with other entities. Here we note that every recipient is unique.

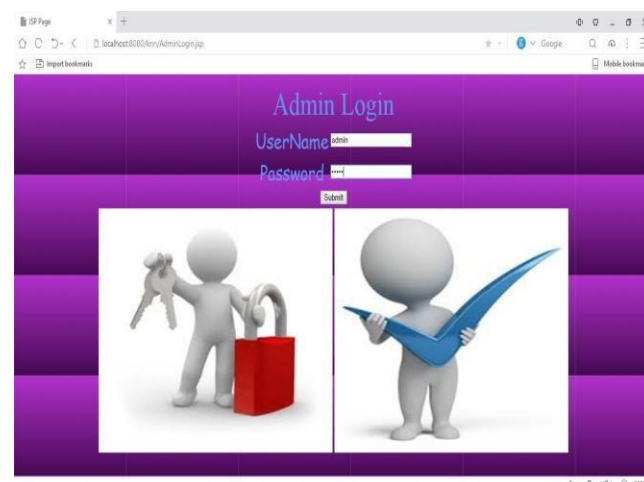
CLOUD SERVER MODULE

A cloud server provides a platform for storing encrypted data. Any recipient can receive the encrypted data stored on the cloud server. The cloud server receives the encrypted data from the purifier and sends it to the recipient without performing any computation operations. A cloud server can act maliciously, for example by deleting encrypted data. Whether the cloud server is

interested or not does not affect the security of SACS.

RESULT AND DISCUSSION:

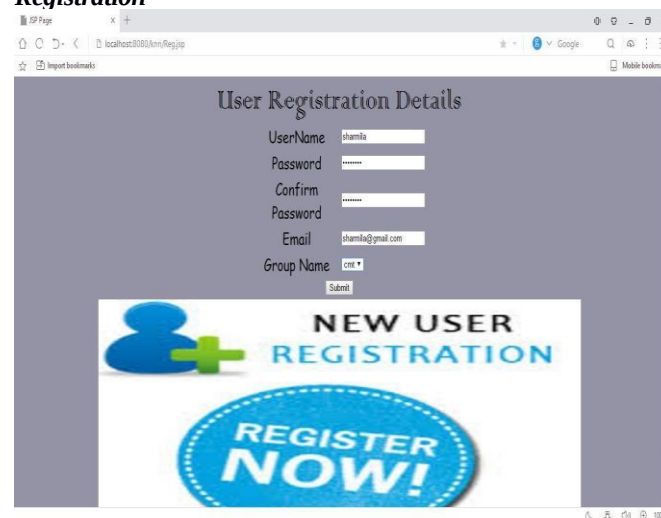
Admin Login



User Login



Registration



CONCLUSION:

In today's modern technology life, it is difficult to protect your data on network and internet without security check, protecting your data from various attacks, especially nowadays as the technology is developing, the attacks are increasing. One of the most important cloud computing issues is security. Users don't trust enough to upload their data to network clouds. In this document, we have provided a series of solutions to mitigate the security issues that exist in Cloud Internet. Infrastructure

FUTURE ENHANCEMENT:

Cloud computing is not fully mature and there is still much to explore. After completing our current study, we can say that security is the most important threat to both cloud computing customers and providers. IT security researchers and experts are working on privacy issues in cloud computing. Various techniques and models have been proposed, but to no avail. When we researched cloud computing security issues, we found that there are no security standards for secure cloud computing. In our future work, we will explore security standards for secure cloud computing.

REFERENCES

- [1] M. T. Beck, J. Camenisch, D. Derler, S. Krenn, H. C. Pohls, K. Samelin, " and D. Slamanig, "Practical strongly invisible and strongly accountable sanitizable signatures," in ACISP 2017, ser. LNCS, vol. 10342, 2017, pp. 437–452.
- [2] J. Camenisch, D. Derler, S. Krenn, H. C. Pohls, K. Samelin, and D. Slamanig, "Chameleon-hashes with ephemeral trapdoors - and applications to invisible sanitizable signatures," in PKC 2017, ser. LNCS, vol. 10175, 2017, pp. 152–182.
- [3] N. Fleischhacker, J. Krupp, G. Malavolta, J. Schneider, D. Schroder, and M. Simkin, "Efficient unlinkable sanitizable signatures from signatures with re-randomizable keys," in PKC 2016, ser. LNCS, vol. 9614, 2016, pp. 301–330.
- [4] I. Damgard, H. Haagh, and C. Orlandi, "Access control encryption: Enforcing information flow with cryptography," in TCC 2016, ser. LNCS, vol. 9986, 2016, pp. 547–576.
- [5] V. Fehr and M. Fischlin, "Sanitizable signcryption: Sanitization over encrypted data (full version)," IACR Cryptology ePrint Archive, vol. 2015, p. 765, 2015.
- [6] R. W. F. Lai, T. Zhang, S. S. M. Chow, and D. Schroder, "Efficient sanitizable signatures without random oracles," in ESORICS 2016, ser. LNCS, vol. 9878, 2016, pp. 363–380.

D.CERTIFICATE

