ABSTRACT

Cloud computing is the practice of using a network of remote servers that are used to store, manage and process data, rather than a local server or a personal computer. Now a days, cloud computing is one of the dominant methods used for providing computing infrastructure for Internet services. Actually, most cloud computing concepts originally came from the academic research community, but as clouds grew in popularity, industry has driven their design and development. Cloud computing offers computing and software services on demand by connecting to computing resources and access to IT managed services with an ease. This flexibility of cloud based services comes with risk of security and privacy of user’s data. Client privacy is a tentative issue as all clients do not have the same demands regarding privacy.

In this project, the proposed system provides the cloud data security using the homomorphic encryption technique. This technique provides functionality to perform operations on encrypted data without using the private key and without decrypting that data. After decrypting the result of this operation, it is the same as if we carried out the calculation on the plain data. This major strength of homomorphic encryption allows the cloud service providers to perform operations on encrypted client data without compromising on the client data privacy. In most real world applications, securing cloud data with efficient homomorphic encryption systems is a work in progress. However, there are a number of practical applications where we use these systems and there is scope for designing more efficient homomorphic encryption systems in the future.
TABLE OF CONTENTS

Abstract ........................................................................................................................................ ii

Table of Contents ......................................................................................................................... iii

List of Figures ............................................................................................................................... vi

List of Tables ................................................................................................................................. viii

1. Background and Rationale .................................................................................................... 1

   1.1 Importance of Client Privacy in Cloud Computing ......................................................... 1

   1.2 Introduction to Homomorphic Encryption ....................................................................... 1

   1.3 Partially Homomorphic Encryption Systems ................................................................. 2

       1.3.1 Additive Homomorphic Encryption Systems ........................................................... 2

       1.3.2 Multiplicative Homomorphic Encryption Systems .................................................. 3

       1.3.3 Additive and Multiplicative Homomorphic Encryption Systems ............................ 3

   1.4 Fully Homomorphic Encryption Systems ........................................................................ 3

2. Narrative .................................................................................................................................. 4

   2.1 Paillier’s Algorithm ......................................................................................................... 4

   2.2 Problem Statement .......................................................................................................... 4

   2.3 Motivation ....................................................................................................................... 5

   2.4 Project Objective ............................................................................................................. 6

   2.4 Literature Review .......................................................................................................... 6

3. Proposed System Design ...................................................................................................... 8

   3.1 Framework .................................................................................................................... 9

   3.2 Stepwise Procedure for Project Development .............................................................. 9

   3.3 Microsoft Azure ........................................................................................................... 10
3.4 Environment .............................................................................................................11

3.4.1 Eclipse IDE .......................................................................................................12

3.4.2 J2EE (Java 2 Enterprise Edition) ....................................................................13

3.4.3 Java Servlets ....................................................................................................13

3.4.4 JSP Technology ..............................................................................................13

3.5 Software Requirements .......................................................................................14

3.6 Proposed System Architecture ..........................................................................15

3.7 Methodology .......................................................................................................16

3.8 Microsoft Azure ................................................................................................17

4. Implementation and Results ................................................................................17

4.1 Application Deployment using Azure .................................................................17

4.2 User Interface .....................................................................................................20

4.3 First Time Login ..................................................................................................20

4.4 Login Screen ......................................................................................................21

4.5 File Upload Mechanism ......................................................................................23

4.6 Encryption Mechanism ......................................................................................24

4.6.1 Download Encrypted File .............................................................................26

4.7 Decryption Mechanism ......................................................................................29

4.8 Encrypting and Decrypting Image Files .............................................................30

4.9 Encrypting and Decrypting Texts Files ..............................................................33

4.10 Results ................................................................................................................34

5. Testing and Evaluation .........................................................................................35

5.1 Types of Testing ................................................................................................35
LIST OF FIGURES

Figure 1: Proposed System Design ......................................................................................9
Figure 2: Architecture of the Proposed System .................................................................15
Figure 3: Homomorphic Encryption Applied to Cloud Computing Security .................16
Figure 4: Eclipse RunInEmulator.cmd Tool .................................................................17
Figure 5: Command Prompt Showing Azure Storage Emulator .......................................18
Figure 6: Start of Azure Deployment Process .................................................................19
Figure 7: Run of Azure Deployment Process .................................................................19
Figure 8: User Interface .....................................................................................................20
Figure 9: Registration Form ...............................................................................................21
Figure 10: Login Screen ....................................................................................................22
Figure 11: File Upload Screen ...........................................................................................23
Figure 12: Input File Containing Numeric Data ...............................................................23
Figure 13: Homomorphic Encryption Mechanism ............................................................24
Figure 14: Key or Password Generated during Encryption .............................................25
Figure 15: Encryption completes Successfully ...............................................................25
Figure 16: Download Encrypted File ...............................................................................26
Figure 17: Download File using Key or Password ..........................................................27
Figure 18: Download Encrypted File after Password Verification ...................................27
Figure 19: Opening the Downloaded .txt File ...............................................................28
Figure 20: File Showing Encrypted Values of Input in BigInteger Format .................28
Figure 21: Decryption Mechanism ....................................................................................29
Figure 22: Decrypted Final Output File ............................................................................30
Figure 23: File with Encrypted Zeros .................................................................31
Figure 24: Image File to be Encrypted .................................................................31
Figure 25: Image File after Decryption .................................................................33
Figure 26: Input Text File to be Encrypted ..........................................................34
Figure 27: Decrypted Output File with Original Text Data ...............................34
Figure 28: Authorized User Successfully Logged into the Website .................38
Figure 29: Encrypting Input File ..........................................................................39
Figure 30: Encryption Process Completes ..........................................................40
Figure 31: Encrypted Input Data in BigInteger Format .....................................41
Figure 32: User Selects Decrypt Process ..............................................................42
Figure 33: File gets Downloaded to Local Machine ............................................42
Figure 34: Output File has Correct Data (Sum of Input) after Decryption .......43
Figure 35: Input File with Negative Integer .........................................................43
Figure 36: Output File when Negative Integers were Used as Input ...............44
Figure 37: Input File with Large Integers ............................................................44
Figure 38: Output File when Large Integers Used as Input ...............................45
Figure 39: File with ‘0’ as Input .........................................................................45
Figure 40: Output File when ‘0’ was Used as Input ............................................46
Figure 41: Input File with Text Data ....................................................................46
Figure 42: Output File when Text Data was Used as Input ..............................47
Figure 43: Input Image File ..................................................................................47
Figure 44: Encrypted Image File Shows no Data ...............................................48
Figure 45: Decrypted Output File same as Original Input Image ......................48
Figure 46: Empty Input File........................................................................................................49

Figure 47: Display Error Message to User ..................................................................................49

LIST OF TABLES

Table 1: Comparison of Encryption Techniques........................................................................7
1. BACKGROUND AND RATIONALE

1.1 Importance of Client Privacy in Cloud Computing

In cloud computing, small companies or clients use services such as computational capabilities of cloud service providers. Sometimes the computationally strong client service provider servers cannot be trusted as clients do not exhibit full control over them. Hence cloud computing security is challenging and is an area of active research. Any organization that is considering a move to the cloud will most importantly focus on cloud security [13]. It must make sure that the client data is secured and client private information is kept confidential at all times. This implies that the major challenges that any cloud computing service provider must overcome are to make sure that if its servers are attacked by hackers, the client data cannot be stolen or misused and secondly the confidential client data must remain invisible even to the cloud service providers. The data needs to be encrypted and then sent to the cloud provider. This means that the provider cannot operate on the data until it decrypts the same. The client has to provide private key to the server (has to compromise on confidentiality) for decryption and only then subsequent operations can be done on the data.

1.2 Introduction to Homomorphic Encryption

Homomorphic encryption technique applied on the cloud computing security provides functionality to overcome the above mentioned security challenges. Using this technique, operations can be performed on encrypted data without the need for decryption. Once operations are performed on client data by the cloud provider the client
can decrypt the result achieving its goal without the cloud provider server knowing anything about the data it operated on.

The concept of homomorphic encryption was first discussed by Rivest, Adleman and Dertouzos [2]. They used some restricted classes of functions such as addition or multiplication in their discussion. Fully Homomorphic Encryption (FHE) methods that were recently developed have the capability to support all kinds of functions.

The first FHE system was developed by Craig Gentry [4] but his system was very complex. Zvika Brakerski and Vinod Vaikuntanathan [5] gave a simpler and also efficient FHE. FHE systems continues to be an active area of research especially given the popularity and dominance of cloud computing.

Homomorphic Encryption systems can be classified into different types based on the operations that it allows on its raw data. Following sections discuss these different types of homomorphic encryption systems.

1.3 Partially Homomorphic Encryption Systems

Partially homomorphic encryption systems can support only a single operation on encrypted data as they are defined over a group. Different types of partially homomorphic Encryption systems are as follows.

1.3.1 Additive Homomorphic Encryption Systems

This type of system allows homomorphic computation of only the addition operation. Some well-known examples of additive homomorphic encryption systems include Goldwasser-Micali system [6], Paillier system [7] and Damgard-Jurink’s
generalization of Paillier system [8]. These subsystems satisfy the property that the product of two cipher texts will decrypt to the sum of their plain texts.

\[ \text{Enc}(x + y) = \text{Enc}(x) \cdot \text{Enc}(y) \] [1]

### 1.3.2 Multiplicative Homomorphic Encryption Systems

This type of system allows homomorphic computation of only the Multiplication operation. Some well-known examples of multiplicative homomorphic encryption systems are RSA algorithm [9] and ElGamal encryption system [10]. These subsystems satisfy the property that the product of the cipher texts equals the cipher of the product.

\[ \text{Enc}(x \cdot y) = \text{Enc}(x) \cdot \text{Enc}(y) \] [1]

### 1.3.3 Additive and Multiplicative Homomorphic Encryption Systems

These systems allow arbitrary many homomorphic computations of one type and limited number of operations of the other type i.e. it allows both addition and multiplication operations but is not fully homomorphic. An example of this kind would be Boneh-Goh-Nissim cryptosystem [11]. It supports computation of an unlimited number of additions but at most one multiplication.

### 1.4 Fully Homomorphic Encryption Systems

These systems allow arbitrary number of additions and multiplications and thus many types of computations can be done on the encrypted data that is stored in the cloud without the need for any decryption. This means that operations on confidential data can now be outsourced to the cloud server keeping the secret key that can decrypt the result of the operation.
2. NARRATIVE

2.1 Paillier Algorithm

The Paillier cryptosystem, named after and invented by Pascal Paillier in 1999, is a probabilistic asymmetric algorithm for public key cryptography [7]. The problem of computing n-th residue classes is believed to be computationally difficult [7]. The decisional composite residuosity assumption is the intractability hypothesis upon which this cryptosystem is based [7]. The Paillier [14] scheme is an additive homomorphic cryptosystem; this means that, given only the public-key and the encryption of \( m_1 \) and \( m_2 \), one can compute the encryption of \( m_1 + m_2 \).

2.2 Problem Statement

The major challenges that any cloud computing service provider must overcome are to make sure that if its servers are attacked by hackers, the client data cannot be stolen or misused and secondly the confidential client data must remain invisible even to the cloud service providers. The data needs to be encrypted and then sent to the cloud provider. This means that the provider cannot operate on the data until it decrypts the same. This becomes a problem as the client has to compromise on confidentiality and share the private key to the server for decryption and only then subsequent operations can be done on the data. Also, this decryption process can be a performance issue for the service provider whenever it wants to operate on its client data.
2.3 Motivation

Cloud service providers have to make sure that their client data is kept confidential at all times. If these service providers suddenly go out of business due to some reason (e.g. bankruptcy) or if there are security attacks done by hackers on the cloud service provider servers, it must be made sure that client data cannot be stolen or misused in any way. There have been a number of cases in recent times where hackers exploited the loopholes and security vulnerabilities in the cloud applications and misused important client data. Below are some of the recent attacks on cloud.

a) September 2014 - Apple's iCloud online storage service was hacked [15].

b) June 2014 - The popular online note-taking service Evernote’s Cloud service brought down by denial-of-service attack [16].

c) June 2014 - Code Spaces, another code-hosting company and software collaboration platform goes dark after its Amazon Web Services cloud service EC2 was hacked[17].

As cloud computing and its applications have become so important and popular, there is a huge need for security techniques that can not only encrypt client data on the cloud but also operate on encrypted data without the need for decryption. Once operations are performed on client data by the cloud provider the client can decrypt the result achieving its goal without the cloud provider server knowing anything about the data it operated on.
2.4 Project Objective

The main objective of this project is to implement a cloud based application which provides data security to the users/clients using homomorphic encryption technique. This application will be a Cloud based Open Share point Website from where the users can store and manage their data onto the cloud. This data can be application data, personal data or computed data and will be in encrypted format.

2.5 Literature Review

Cryptography is very important feature in the field of information technology and used to protect data privacy using several standard methods. Cloud computing, one of the dominant methods where computing resources are leased on demand, makes use of several data encryption techniques to secure the cloud servers. Different types of existing encryption techniques must be analyzed on the basis of performance and security. Cloud users must select one of these techniques in order to protect and verify their sensitive data.

Basically, there are two types of encryption techniques [18], symmetric encryption (also known as secret key encryption) and asymmetric encryption (also known as public key encryption). Symmetric encryption is the best known technique and perhaps the oldest. A secret key is applied to the text of a message to change its content in a specific manner way. The main problem with secret keys is transmitting them over the Internet or a wide network and at the same time making sure it does not end in the wrong hands. This problem can be resolved with asymmetric encryption, where two related keys, a public key which is freely available to anyone who wants to send a message and a
private key which is kept secret is used for decryption [19], so that only the intended recipient or the user know it. Having said that, asymmetric encryption techniques are slower than symmetric techniques, because they require a lot of computational processing power [20] as public key encryption is based on computationally intensive mathematical functions. As security is the main concern for cloud computing environment, asymmetric encryption techniques can be chosen over symmetric and hence performance of the system is slightly compromised.

Homomorphic encryption is an asymmetric technique applied on the cloud computing security. With this technique, operations can be performed on encrypted data without the need for decryption. Once operations are performed on client data by the cloud provider the client can decrypt the result achieving its goal without the cloud provider server knowing anything about the data it operated on. RSA algorithm [9] or Paillier’s algorithm [7] can be used to implement the homomorphic encryption.

Table 1 below shows the performance analysis and comparison of different symmetric and asymmetric encryption techniques that can be used for cloud security.

<table>
<thead>
<tr>
<th>Property</th>
<th>Symmetric</th>
<th>Asymmetric</th>
</tr>
</thead>
<tbody>
<tr>
<td>Types</td>
<td>DES, TDES, AES</td>
<td>RSA, Paillier Algorithms</td>
</tr>
<tr>
<td>Key Distribution</td>
<td>Difficult</td>
<td>Easy</td>
</tr>
<tr>
<td>Security</td>
<td>Moderate</td>
<td>Highest</td>
</tr>
<tr>
<td>Secure Services</td>
<td>Confidentiality</td>
<td>Confidentiality, integrity, non-repudiation</td>
</tr>
<tr>
<td>Encryption</td>
<td>Faster</td>
<td>Slow</td>
</tr>
<tr>
<td>Decryption</td>
<td>Faster</td>
<td>Slow</td>
</tr>
<tr>
<td>Complexity</td>
<td>O(Log N) - DES</td>
<td>O(N3) - RSA</td>
</tr>
</tbody>
</table>

Table 1: Comparison of encryption techniques
3. PROPOSED SYSTEM DESIGN

The proposed system is cloud based platform which will be used by the user to upload data onto the cloud. An open share point website will be deployed onto the cloud for this purpose. Once logged in successfully, the user will be prompted to encrypt the data before moving it onto the cloud and a private key will be generated. This key will be used when the user decrypts final output file/data. A folder will be created in the database for each authorized user after the first successful login which will contain all files/data for that user in homomorphic encryption format.

Figure 1 shown below contains different modules that are part of the proposed system design. These modules are the data upload module, encryption module, decryption module and the result module. The input is a .txt file containing data that needs to be uploaded and stored on to the cloud with the help of data upload module. The data stored is then encrypted and a secret key is also generated for the decryption process. Operations such as addition will be performed on the encrypted data stored onto the cloud server and the final result will be decrypted using the secret key in the decryption module. Result module displays the final result as a text file. The detailed functionality of all these modules will be discussed in the later sections.
3.1 Framework

Asymmetric encryption technique such as Paillier’s Algorithm is used to implement and demonstrate the property of homomorphism in this application. Paillier’s algorithm realizes the property of additive homomorphic encryption since the product of two cipher texts will decrypt to the sum of their corresponding plain texts. We use this technique because operations can be performed on the encrypted data without the need for decryption.

3.2 Stepwise Procedure for Project Development

The following steps are involved in the development of this application to protect the confidentiality and integrity of cloud data using Homomorphic encryption:
1. Study and analyze different encryption techniques that can be used to secure cloud data.

2. Design a web based application that can be deployed on the cloud and can provide functionality to secure user data.

3. Develop the web application on MVC architecture using Java 2 enterprise edition. This can be used to collect user data which is then processed through Java servlets and also Java servlet packages.

4. In order to build the data tables and store information such as authentication details, user data uploaded on to the cloud etc, MySQL enterprise data software is used.

5. Homomorphic Encryption and decryption techniques are implemented using the Paillier’s algorithm using the security packages provided by Java. These packages are also used for creation of the secrets keys.

6. Microsoft azure software is used to deploy the web application. The following section explains the features, functionality and uses of this software.

### 3.3 Microsoft Azure

Microsoft Azure is a cloud computing platform created by Microsoft to build, deploy and manage applications such as a sharepoint website created in this project. Microsoft’s Azure Services Platform is a group of cloud technologies, each providing a specific set of services to the application developers. The Azure Services Platform can be used both by applications running in the cloud and by applications running on local systems. Core services provided by the Azure platform:
• Hardware infrastructure and maintenance
• Technology platform
• Scalability
• Security
• Disaster recovery and backups
• Monitoring

In this project we use Azure Toolkit for Eclipse with Java (by Microsoft Open Technologies) for Java to create, develop, test, and deploy our Azure application using the Eclipse development environment. It is an Open Source project, whose source code is available under the Apache License 2.0 from the project’s site at [12]. This toolkit is installed and a small Hello world Azure application is created and tested in Eclipse. We first package the web application and then run and deploy it in the Microsoft Azure Compute Emulator from the command line. The emulator provides a local simulation of the Azure compute services so that the application can be tested in the local development environment before deploying it to Azure. Debugging of this Azure application can also be done with the help of Eclipse.

3.4 Environment

The proposed system is implemented in Java and the environment used is Eclipse IDE.
3.4.1 Eclipse IDE

Eclipse is an integrated development environment (IDE). It comprises of a workspace and an extensible plug-in system. This IDE can be used to develop applications coded in Java. When certain plug-ins are added to the Eclipse IDE, we can then use it for making applications in C, C++, COBOL, JavaScript, Perl, PHP, Python, and Ruby and many other programming languages. Eclipse Java development tools (JDT) is one of the development environment provided by Eclipse for Java and scala. Eclipse CDTis provided for C/C++ and Eclipse PDTis the environment for PHP. The Java development tools are present in Eclipse software development kit (SDK) for developers. A built-in incremental Java compiler is provided in this IDE and a full model of the Java source files. This allows for advanced refactoring techniques and code analysis. Users can enhance its capabilities by installing plug-ins coded for the Eclipse Platform, such as development toolkits for other programming languages, and can write and contribute their own plug-in modules. Eclipse SDK was released under the terms of the Eclipse Public License; it is free and open source software.

3.4.2 J2EE (Java 2 Enterprise Edition)

J2EE stands for Java 2 Platform Enterprise Edition, it is a platform-independent, Java-centric environment from Sun. It is used for developing, building and deploying of online Web-based enterprise applications. The J2EE platform comprises of a set of services, APIs, and protocols that can be used for developing web based applications.
3.4.3 Java Servlets

Java Servlets are java programs written at server side. Web container of application server provides runtime environment for deploying java servlets. When the J2EE application server gets a client request, servlets are executed. Some of the features of Java Servlets are:

1) Security: Java Servlets inherits the security feature that the Web container provides.

2) Session Management: End user identity and state is kept intact across more than one requests.

3) Instance persistence: Frequent disk access is prevented. This enhances server performance.

3.4.4 JSP Technology

Work profiles of a Web designer and a Web developer are brought together with the help of JSP technology. HTML can be used by the web designer to design and the layout for a Web page. Then the Web developer can use Java code and other JSP related tags and work independently to write the business logic. Files are tied up by servlets to handle the static presentation logic and the dynamic business logic independently. Also, Java can be embedded directly into an HTML page with the help of JSP by using special tags. Extensive coding is involved in Servlet programming. Static code content and dynamic code content has to be identified and separated if any changes need to be made.
to the code to ease incorporation of the changes. This also allows both Web developers and the Web designer to work independently.

3.5 Software Requirements

The following are the basic resources required in developing and executing the project:

Hardware Requirements

Processor Type : Intel Core i3  
Speed : 1.8 GHZ  
Ram : 4 GB MB  
Hard disk : 40 GB HD  
Keyboard : Standard Windows Keyboard  
Mouse : Two or Three Button Mouse

Software Requirements

Operating System : Windows 7 / Windows 8  
Programming Package : JAVA / J2EE  
IDE Tools : Eclipse.  
SDK : JDK1.5.0  
Backend : MySql  
Server : Tomcat7.0
3.6 Proposed System Architecture

A high level proposed architecture of the system is as shown in Figure 2 below. All the various components and their communication with each other are clearly shown here. External users are those who have authorization to login to the website to store and manage their data. They communicate with the web browser to move data on to the cloud. The web browser connects with the application server which is responsible for efficient execution of programs, routines, scripts that support this application. This will execute the homomorphic algorithms needed to secure the user’s data. Data is moved through the application server to the database. Also, whenever a user requests for a file, the database is queried appropriately to extract the data and display the same on the website via the application server.

Figure 2. Architecture of the Proposed System
3.7 Methodology

The main purpose of this application is to provide functionality of performing operations on encrypted data. In this application, we are performing operations such as addition on encrypted data to implement Paillier’s Homomorphic encryption technique; we make use of integer values present in a text file as our input. Following figure illustrates the methodology that is employed to implement this web application. It also shows the flow of data from the client company to the cloud service provider and vice versa.

Figure 3: Homomorphic Encryption Applied to Cloud Computing Security [13].
4. IMPLEMENTATION AND RESULTS

4.1 Application Deployment using Azure

The web application that is built in this project needs to be deployed on to the cloud environment using Microsoft Azure. Figure 4 clearly shows the deployment process done with the help of Eclipse RunInEmulator.cmd tool.

Figure 4: Eclipse RunInEmulator.cmd tool

After the runInEmulator.cmd command is executed through Eclipse, it opens the command prompt showing the windows Azure storage emulator tool 3.3.0.0. This is shown in Figure 5.
Figure 5: Command Prompt showing Azure Storage Emulator

Azure Deployment Project will then be opened in the Microsoft Azure Compute emulator which will start the deployment process. Figure 6 and Figure 7 show the workerRole1 file that starts and executes the deployment process respectively.
Figure 6: Start of Azure Deployment Process

Figure 7: Run of Azure Deployment Process
4.2 User Interface

User interface consists of various functionalities such as user authorization, uploading of files to the cloud, encryption mechanism, decryption mechanism and download mechanism to view the decrypted files. Figure 8 shows the user interface.

![User Interface](image)

Figure 8: User interface

4.3 First Time Login

First time login can be done only after the registration process is successfully done. The user must click on the register button which opens up a form. All the fields need to be filled out appropriately. Validation is done for all the fields and if any field has incorrect data, an error message is displayed for the user. Figure 9 shows the registration form.
4.4 Login Screen

After the registration, the user can login to the sharepoint site using the appropriate credentials. The login information is verified using the authentication process. Once logged in the user can start managing the data on the cloud server. If invalid login information is entered, an error message is shown on the website. Figure 10 shows the login screen.
4.5 File Upload Mechanism

Using this mechanism user can upload data onto the website for the local machine. Single file upload button can used to upload a single file and similarly if the user wishes to upload multiple file at the same time, multiple file upload button can be used. Figure 11 and Figure 12 show this functionality.

Input file upload is info.txt. This file contains numeric data which will help to prove the additive homomorphic property using Paillier’s algorithm. The encryption and decryption mechanism is shown in following sections.
Figure 11: File Upload Screen

Figure 12: Input file containing Numeric data
4.6 Encryption Mechanism

After the upload process completes successfully, user is directed to the encryption page as shown in Figure 13. Here homomorphic encryption is applied on the file containing integer data using the Paillier algorithm. While the file gets encrypted, there is also functionality to generate a key or a password which can be used while decrypting the same file. This is shown in Figure 14. Figure 15 shows that the encryption process completed successfully.

![Figure 13: Homomorphic Encryption Mechanism](image-url)
Figure 14: Key or Password Generated during Encryption

Figure 15: Encryption completes Successfully
4.6.1 Download Encrypted File

After the encryption process completes, user can download the encrypted file to view its contents. Figure 16, Figure 17, Figure 18, Figure 19 and Figure 20 show the various steps involved in the download process.

![Figure 16: Download Encrypted file](image-url)
Figure 17: Download file using Key or Password

Figure 18: Download Encrypted file after Password Verification
Figure 19: Opening the downloaded .txt file

Figure 20: File showing encrypted values of input in biginteger format.

28
4.7 Decryption Mechanism

After the encryption process completes successfully, multiplication operation is done on the cipher texts in order to demonstrate the Paillier’s algorithm implementation which satisfies the additive homomorphic property. The final result of the operation when decrypted will produce the same result as if addition operation was done on plain text. This proves the homomorphic property. This can be viewed in the output file shown below in Figure 22. The password or key generated during the encryption process has to be keyed in before starting the decryption. The output file can be downloaded using the download file functionality shown in Figure 21 after the decryption process completes successfully.

Figure 21: Decryption Mechanism
4.8 Encrypting and Decrypting Image Files

Image files consist of digital data that is stored in the form of pixels. Each pixel has a number of bits that represent its color. As we know that Paillier’s algorithm i.e. the additive homomorphic property works only on integer data, we cannot use encrypted images for multiplication and hence cannot generate any meaningful output.

In order to execute the Paillier’s algorithm and homomorphic encryption on images they must be first converted into integer type data. The following steps show that an image can be converted into integer type data using its pixel information.

1. Initially, image to be encrypted and a file of encrypted zeros are taken as input as shown in Figure 23. Image to be encrypted is shown in Figure 24.
2. The image is then processed with the help of java.awt.Image package to form pixels so that pixel values can be extracted.

3. Once pixels are formed, each pixel is looped over as per the height and
width of the image and encrypted with the help of Paillier’s algorithms along with encrypted zeros.

4. An array of big integers is formed whose length is equal to height and width of the image.

5. With the help of Paillier’s algorithm’s mod operation, each value in this array is formed by multiplying with one constant encrypted zero which yields in the sum of pixel value with encrypted zero.

6. The result generated is encrypted image array containing the position of pixels which are encrypted and hence we can work with this encrypted pixels.

7. After encrypted array is looped over the entire height and width of the image, this value is passed into Hough accumulate function to get map value which is the applied mod function to operate on the encrypted pixel values.

8. This Hash map will be used again to decrypt the image using homomorphism technique.

9. The decryption process starts with encrypted big integer array.

10. Using Paillier’s algorithm, loop over each value present in the encrypted array and then decrypting each value at the particular looping point with the help of Paillier’s algorithm decryption technique.

11. The result will return hash set value of the pixel which is also the decrypted value by applying homomorphism.

12. Hence homomorphic encryption is applied using Paillier’s algorithm. Following Figure 25 shows the original image after decryption.
4.9 Encrypting and Decrypting Text Files

As we know that Paillier’s algorithm i.e. the additive homomorphic property works only on integer data, we cannot use text other than integers. Also, the product of cipher text which equals to the sum of plain text will not provide any meaning with text data other than integers. Hence, if we try to execute the Paillier’s algorithm on text, we will end up with an output which contains junk data.

One way to implement the homomorphism with text would be to convert the text into Big integers and then encrypt. Multiplication can be done with encrypted value of zero so that when we decrypt the final results it will the sum of original Big integer added with zero. This will make sure that we get the original text upon decryption. This can be seen in Figure 26 (input file with text) and Figure 27 which is the decrypted output.
4.10 Results

This implementation of a cloud based web application can provide functionality for users to homomorphically encrypt data on the cloud servers. The main advantage of this technique is that operations such as multiplication can be done on encrypted data without the need for decryption. The user’s data is kept confidentiality as the cloud server that operates on it does not know what data it operated upon. Also, if the cloud service
provider servers are hacked by malicious attackers, the user’s data is secured and cannot be misused as it is homomorphic encrypted. Additional functionality was provided in this web application to encrypt image and text files.

5. TESTING AND EVALUATION

Testing plays an important role for any application to detect errors. It is the process of discovering errors, which are mostly probable weakness or fault of the proposed system. The functionality of various components, assemblies and final products can be verified through testing. Testing process makes sure that the software of the system meets the requirements based on the user expectations. It also verifies for the system’s case of failing in any unacceptable way.

5.1 Types of Testing

There are different types of testing each representing a specific testing requirement. They are briefly described in sections below.

5.1.1 Unit Testing

The process of unit testing involves validation of the internal programming logic’s functionality, along with the program input and the corresponding output. The validation also involves the internal code flow and all the decision branches. Unit testing involves the testing of individual software applications, which is done before integration. It is a structural testing, which relies on the knowledge of construction and its invasion.
Unit testing ensures a unique path for each business process to perform specifically for the documented specifications, which defines clear inputs and the expected results.

5.1.2 Integration Testing

Integration testing is performed for testing integrated software components to check if they are running as one program. The testing process done in unit testing shows the successful results with individual components, whereas in integration testing, the correctness and consistency of the combination of components is checked. Thus, this type of testing is mainly done to check the problems that rise with the combination of different components.

5.1.3 Functional Testing

In functional testing, the functionality of specified functions demonstrated based on some technical and business requirements, user manuals and system documentation. Functional testing mainly concentrates on valid and invalid inputs, functions of the system followed by the output. Besides these, functional testing also focuses on key functions, various requirements, few special test cases, predefined and successive processes, data fields and business process flows. Before performing the functional testing, additional test are identified and the corresponding effective values of the current test are considered.
5.1.4 System Testing

System testing is performed so as to check if the system meets all the mentioned requirements. It also tests for ensuring the configuration of known and predictable results. Configuration oriented system integration test is considered as an example for system testing. This type of testing is based on integration points, process description and flows and on emphasizing pre-driven process links. System testing provides various inputs along with the output without considering the working structure of the software.

5.2 Test Case 1

In this case we verify whether the Paillier’s additive homomorphic property is getting implemented or not i.e. the product of the cipher text is equal to the sum of the plain texts.

Login in to the Share point site. Evaluate if the valid user is able to login in to the website and if not an error message will be displayed. Figure 28 shows that user is able to login to the website.
Figure 28: Authorized user successfully logged in to the Website
After Successful login user can upload the files on to share point site and will be provided with an option to encrypt these files with homomorphic encryption. Verify whether the encryption process completes or not.

Figure 29: Encrypting input file
Test for Encryption. Evaluate if encryption is successful then file must be uploaded to the cloud database otherwise an appropriate error will be displayed. In order to do the file has to be downloaded and checked for proper encryption. The downloaded file is shown in the below Figure 24.
Figure 31: Encrypted Input data in biginteger format

Decrypting and downloading files to the local system. Evaluate if user selects to download a particular file then again Homomorphic decryption process occurs and the file gets downloaded successfully with correct output.
Figure 32: User selects decrypt process

Figure 33: File gets downloaded to local machine
5.3 Test Case 2 – Negative Integers as Input

To verify the Paillier’s additive homomorphic property for negative integers.

Input file shown below in Figure 30 contains negative integer as -100.
After encryption and decryption process gets completed, the output file in Figure 31 shows that even with a negative integer as input, Pailler’s algorithm is successfully implemented.

5.4 Test Case 3 – Large Integers as Input

To verify the Paillier’s additive homomorphic property for large integers.

Input file shown below in Figure 32 contains large integers.
Figure 38 shows that Paillier’s additive homomorphic property is successfully executed for large integers.

![Output File when Large Integers used as Input](image1)

**Figure 38: Output File when Large Integers used as Input**

### 5.5 Test Case 4 – ‘0’ as Input

To verify the Paillier’s additive homomorphic property with ‘0’ as input.

Input file shown below in Figure 39 contains large integers.

![File with 0 as Input](image2)

**Figure 39: File with 0 as Input**
Figure 40 shows that Paillier’s additive homomorphic property is successfully executed when 0 was used as input integer.

![Figure 40: Output File when ‘0’ was used as Input](image)

**5.6 Test Case 5 – Text Data**

To verify the Paillier’s additive homomorphic property with text data.

Input file shown below in Figure 41 contains text data.

![Figure 41: Input File with Text Data](image)

Figure 42 shows that the final output after decrypting the text data is same as the input file.
5.7 Test Case 6 – Image File

To verify the Paillier’s additive homomorphic property with image file.

Input file shown below in Figure 43 is a.jpg image file.
Figure 44 shows that the file cannot be viewed when homomorphically encrypted.

Figure 44: Encrypted Image File shows no data

Figure 45 shows that the output file is the same image as the input file.

Figure 45: Decrypted Output File same as Original Input Image
5.7 Test Case 7 – Empty Input File

Figure 46 shows that the input file is empty with no data.

Figure 46: Empty Input File

Figure 47 shows that the encryption process will fail with the appropriate error message for the user to provide valid input file.

Figure 47: Display Error Message to User
6. CONCLUSION AND FUTURE WORK

Homomorphic encryption systems have rapidly evolved in the last couple of years. In this project a cloud based web application was implemented to encrypt data on the cloud servers using Paillier’s homomorphic encryption algorithm. The main advantage of this technique is that operations can be performed on encrypted data stored on the cloud without the need for decryption. Hence, user’s data is kept confidentiality as the cloud server that operates on it does not know what data it operated upon. Also, if the cloud service provider servers are hacked by malicious attackers, the user’s data is secured and cannot be misused as it is homomorphic encrypted.

Homomorphic encryption in cloud computing is an active area of research considering the dominance of cloud computing in today’s market place. Hence more work can be done in future to build more efficient cloud computing applications similar to the one that is implemented as part of this project. Additional functionality to work with different types of input files can be implemented for this application in the future.

BIBLIOGRAPHY AND REFERENCES


[12] https://github.com/MSOpenTech/WindowsAzureToolkitForEclipseWithJava


APPENDIX 1

The following code implements Paillier’s Algorithm for integer and text data.

Paillier_ency.java

```java
package cryptography;
import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.math.BigInteger;
import java.util.Random;
import java.util.logging.Level;
import java.util.logging.Logger;
import thep.paillier.EncryptedInteger;
import thep.paillier.PrivateKey;
import thep.paillier.PublicKey;
public class Paillier_ency {
    public BigInteger toBigInteger(String foo) {
        return new BigInteger(foo.getBytes());
    }
    public String fromBigInteger(BigInteger bar) {
        return new String(bar.toByteArray());
    }
    public static void main(String args[]) {
    }
    public String Homomorphic(String folderp,String dirurl, String dest, String filename) throws Exception {
        String str="true";
        String desto=folderp;
        System.out.println("directory url:"+desto);
        System.out.println("directory url:"+dirurl);
        System.out.println("filepath:"+dest);
        System.out.println("filename:"+filename);
        String[] information=new String[2];
        // BigInteger[] info1=new BigInteger[2];
        BigInteger val1;
        BigInteger val2;
        Paillier_ency paillier=new Paillier_ency();
        //BigInteger[] data;
        BufferedReader br = new BufferedReader(new FileReader(dirurl));
    }
}
```
String sCurrentLine;

while ((sCurrentLine = br.readLine()) != null) {
    information = sCurrentLine.split("","");
    // info1=br.readLine().split("","");
}

System.out.println("From PAillier"+information[0].startsWith("ab");
    if(information[0]!=null && !information[0].startsWith("this")
)
{
    val1 = new BigInteger(information[0]);
    val2 = new BigInteger(information[1]);
    System.out.println(val1);
    System.out.println(val2);
    //PrivateKey privkey = new PrivateKey(128);
    // Fetch the corresponding public key
    //PublicKey pubkey = privkey.getPublicKey();

    try {
        PrivateKey privkey = new PrivateKey(128);
        // Fetch the corresponding public key
        PublicKey pubkey = privkey.getPublicKey();
        EncryptedInteger enc_a = new EncryptedInteger(val1, pubkey);
        EncryptedInteger enc_b = new EncryptedInteger(val2, pubkey);
        System.out.println(enc_a.getCipherVal());
        System.out.println(enc_b.getCipherVal());
        EncryptedInteger c =enc_a.add(enc_b);
        // m1= new BigInteger("211557367910783653515556345234260646761800312860446987479492632907729278607"
        //EncryptedInteger a = new EncryptedInteger(enc_a.decrypt(privkey),pubkey);
        System.out.println("C = Enc_pub(a) * Enc_pub(b) = " + c.getCipherVal());
        System.out.println("Dec_priv(C) = " +
        paillier.fromBigInteger(c.decrypt(privkey))+"*****"+c.decrypt(privkey));
        String content="Dec_priv(C) = " +c.decrypt(privkey);
        String decryptpat=folderp+"Dec_"+filename;
        System.out.println("Decrypt file path"+decryptpat);
        File file = new File(decryptpat);
        if (!file.exists()) {
            file.createNewFile();
        }
        FileWriter fw = new
    FileWriter(file.getAbsolutePath());
BufferedWriter bw = new BufferedWriter(fw);
bw.write(content);
bw.close();

} catch (Exception ex) {
Logger.getLogger(Paillier_ency.class.getName()).log(Level.SEVERE, null, ex);
}

    else if (information[0]!=null &&
information[0].startsWith("this") ){
    Paillier_ency obj=new Paillier_ency();
    val1 = obj.toBigInteger(information[0]);
    val2 = new BigInteger("0");
    System.out.println(val1);
    System.out.println(val2);
    //PrivateKey privkey = new PrivateKey(128);
    // PublicKey pubkey = privkey.getPublicKey();
    try {
        PrivateKey privkey = new PrivateKey(128);
        // Fetch the corresponding public key
        PublicKey pubkey = privkey.getPublicKey();
    EncryptedInteger enc_a = new EncryptedInteger(val1, pubkey);
    EncryptedInteger enc_b = new EncryptedInteger(val2, pubkey);
    System.out.println(val1+"\\\\+val2);
    System.out.println("Enc_pub(a) = " +
    enc_a.getCipherVal());
    System.out.println("Enc_pub(b) = " +
    enc_b.getCipherVal());
    EncryptedInteger c =enc_a.add(enc_b);
    // m1= new BigInteger("21155736791078365853555634523426064676180031286044698749742632907729278607"
    );
    //EncryptedInteger a = new
    EncryptedInteger(encrypt(a.decrypt(privkey),pubkey);
    System.out.println("C = Enc_pub(a) * Enc_pub(b) = " +
    c.getCipherVal());
    System.out.println("Dec_priv(C) = " +
paillier.fromBigInteger(c.decrypt(privkey))++"******"+c.decrypt(privkey));
    String content="Dec_priv(C) = " +
obj.fromBigInteger(c.decrypt(privkey));
    String decryptpat=folderp+"Dec_"+filename;
    System.out.println("Decrypt file path"+decryptpat);
    File file = new File(decryptpat);
    if (!file.exists()) {
        file.createNewFile();
    }
The Client.java[21] and HomomorphicServer.java[21] code is used to encrypt and decrypt image files using the Paillier’s homomorphic encryption technique. The code used from the web was customized as per the requirements of this web application.