I, Harika Kosaraju, hereby submit this original work as part of the requirements for the degree of Master of Science in Computer Science.

It is entitled:
Authentication of User in the Cloud Using Homomorphic Encryption

Student's name: Harika Kosaraju

This work and its defense approved by:

Committee chair: Prabir Bhattacharya, Ph.D.

Committee member: Raj Bhatnagar, Ph.D.

Committee member: John Franco, Ph.D.
Authentication of User in the Cloud Using Homomorphic Encryption

A thesis submitted to the

Graduate School

Of the University of Cincinnati

In partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

Harika Kosaraju

School of Computing Sciences and Informatics

Advisor: Dr. Prabir Bhattacharya

March 14, 2013

Bachelor of Science in Computer Science, JNTU, India
Abstract

Data on the cloud has many advantages. When the data is going on the cloud, security will play a major role because the data may fall prey to the intruders. One of the ways an intruder can get access to the data or an application is by getting access to the login credentials of a user. There have been many attacks on the cloud because of the weak authentication. There are many security models for authenticating the user on the cloud. Some of them have failed because of reasons like password leaks or security holes.

In this thesis we propose a new model for secure authentication of users in the cloud. We use homomorphic encryption for providing security. We implement the proposed algorithm and analyze the time taken and compare the performance of our model with that of the other models. We compare other models with the proposed model and see how similar they are and how different they are. We also check which model performs better and provides better security.

In the proposed model, we make sure that there is no chance for revealing the password. During authentication of the user, the credentials of the user are not present in the plaintext. They are encrypted and we use homomorphic property so that they are not revealed during the authentication.
Acknowledgements

I would like to use this opportunity to thank my thesis advisor Prof. Prabir Bhattacharya for his support and patience in guiding me in the thesis. I thank Prof. Raj Bhatnagar and Prof. John Franco for being a part of my committee. I thank Prof. Raj Bhatnagar and Prof. Franco for their valuable inputs and their patience in reading my thesis and recommendations.

I would like to thank my parents for their support for listening to my fears and rants in completing the work. I would like to thank my sister, Poojitha for her support and love. I would like to thank my husband, Prasanth for his support and advice in completing the thesis.
Table of Contents

Abstract............................................................................................................................................. 2
Acknowledgements.......................................................................................................................... 3
List of Algorithms............................................................................................................................. 8
List of Figures................................................................................................................................... 9
List of Tables.................................................................................................................................... 10

1. Introduction.................................................................................................................................... 11
   1.1. Introduction............................................................................................................................... 11
   1.2. Motivation............................................................................................................................... 12
   1.3. Our Contribution and Approach........................................................................................... 15
   1.4. Thesis Organization................................................................................................................ 17

2. Background and Related work....................................................................................................... 18
   2.1. Secure Login Authentication Models...................................................................................... 18
       2.1.1. Authentication with Reverse Turing Test........................................................................... 18
       2.1.2. Login Authentication using Cookies.................................................................................. 22
       2.1.3. Comparison between Authentication with Reverse Turing Test and the Proposed
              Model........................................................................................................................................ 24
   2.2. Lamport Scheme..................................................................................................................... 25
       2.2.1. Security offered by Lamport’s Scheme.............................................................................. 26
2.2.2. Comparison of Lamport model with the proposed model.........................26
2.3. Cloud Authentication model based on Anonymous One time Password...........27
  2.3.1. Performance of Cloud Authentication model.........................................29
  2.3.2. Comparison of Cloud Authentication model and the proposed model.........29
2.4. Encryption..................................................................................................30
  2.4.1. Basic Function.........................................................................................30
  2.4.2. Hashing Encryption................................................................................31
  2.4.3. Symmetric Encryption.............................................................................31
  2.4.4. Asymmetric encryption.........................................................................32
  2.4.5. Homomorphic Encryption......................................................................33
    2.4.5.1. Fully Homomorphic and Partially Homomorphic methods...............35
    2.4.5.2. Advantages of Homomorphic Encryption......................................36
  2.4.6 Homomorphic Encryption Schemes.......................................................36
    2.4.6.1. RSA Algorithm...............................................................................36
    2.4.6.2. Gentry's Algorithm.........................................................................38
    2.4.6.3. Goldwasser-Micali Algorithm.......................................................39

3. **Methodology**...............................................................................................41
  3.1. Paillier's Algorithm ..................................................................................41
  3.2. Cloud Platforms.........................................................................................42
  3.3. Proposed Architecture...............................................................................42
  3.4. Proposed Algorithm..................................................................................44
    3.4.1. Security Offered by the proposed algorithm and architecture.............45
4. **Experiments and Results** .............................................................................................................48

4.1. Efficiency of the proposed architecture ..................................................................................49

4.1.1. Efficiency when the length of the plaintext varies.........................................................49

4.1.2. Efficiency when the lengths of p and q vary.................................................................51

4.1.3. Length of the Plain text vs length of the Cipher text..................................................53

4.2. Comparison of Paillier's with Gentry's and Goldwasser-Micali Algorithm.................. 54

4.2.1. Time taken for computation.................................................................54

4.2.2. Length of the ciphertext.........................................................................................54

4.3. Comparison of Authentication Models with the proposed model.............................55

5. **Conclusion and Future Work** .....................................................................................................57

5.1. Conclusion................................................................................................................................57

5.1.1. Summary of Results..................................................................................................58

5.2. Future Work.....................................................................................................................59

Bibliography..................................................................................................................................60

Appendix........................................................................................................................................65
List of Algorithms

1. Cloud Authentication Based on Anonymous One time Password .............................................. 27
2. RSA Algorithm .................................................................................................................... 36
3. Gentry's Algorithm .......................................................................................................... 38
4. Goldwasser-Micali Algorithm .......................................................................................... 39
5. Paillier's Algorithm .......................................................................................................... 41
6. Proposed Algorithm .......................................................................................................... 44
# List of Figures

1. Cloud Structure............................................................................................................................11
2. Importance of Security in Cloud ..................................................................................................12
3. Password authentication with RTT...............................................................................................21
4. Login Authentication with Cookie...............................................................................................23
5. Performance of Cloud Authentication model...............................................................................29
6. Symmetric encryption/decryption process .................................................................................32
7. Asymmetric encryption/decryption process .............................................................................33
8. Homomorphic Encryption............................................................................................................34
9. Common architecture for cloud services......................................................................................42
10. Proposed Architecture.................................................................................................................43
11. Flowchart for proposed model....................................................................................................47
12. Architecture of the system...........................................................................................................48
13. Efficiency when length of plaintext varies..................................................................................50
14. Efficiency when the length of the key varies............................................................................52
15. Length of plaintext vs length of cipher text...............................................................................53
List of Tables

1. Attacks on cloud caused because of stolen credentials..................................................14
2. Time taken for computation according to length of the plaintext.................................49
3. Time taken for computation according to length of the key........................................51
4. Length of plaintext vs length of ciphertext.................................................................53
Chapter 1

Introduction

1.1. Introduction

A Cloud is a network of computers that outsource services over the network to the users. The services are of various types like Platform as a Service, Software as a Service or Data as a Service. Cloud has lot of potential to unleash but should not be at an expense of security and privacy. There have been several attacks on the cloud as discussed in this section because of the compromised security on credentials of the user for authentication in the cloud. We propose a model which offers better security for login credentials of the users.
The models of the cloud are divided on various variables. They are service models and deployment models.

Service models are categorized as Infrastructure as a service, Platform as a service, Software as a Service, Network as a Service and Data as a Service. These are the various ways in which the cloud can be used. Deployment models can be categorized as public cloud, private cloud, community cloud and hybrid cloud.

1.2. Motivation

In figure 2, when an opinion was taken [30] about the challenges of the cloud, maximum number of the cloud users voted for security.

Figure 2: Importance of Security in Cloud

There are many applications and lot of data on the cloud. They have to be secure and should not
be given access to the third party. We put a lot of data on the cloud and the data has to be secured so that this keeps the third party away from analyzing and mining the data on the cloud [14]. The cloud vendors could be Amazon cloud or Microsoft azure cloud or any other. It is too much risk to give a third party access to the raw data in cloud.

Data on the cloud can be secured in many ways. Some of the ways are:

- By giving very limited access to the user
- By encrypting all the data on the cloud
- By outsourcing the computation on the data without outsourcing the control of the data
- By designing secure cloud systems
- By secure Authentication of the user

According to a survey in 2012 [30], the security was at its low in 2010 and 2011. A long list of attacks and the method of the attacks have been documented in the list [29]. The list contains Google, Linkedin, Yahoo, Zappos, Twitter, Global payments like MasterCard and visa, and interestingly, they all have been hacked by a single Chinese hacker named Zeng.

According to the analytics [30], it is known that during 2010, 4 million accounts were compromised by the hackers. In 2011, it rose to 174 million accounts and if anything 2012 will be worse [30]. All this led to a huge outcry from the victims.

There are several attacks on the cloud because of insufficient security [29]. Listed below in Table 1 are some of the attacks which took place in 2012. These attacks have taken place because of the stolen credentials. The credentials for login have been stolen either because of the application being weak or the credentials being exposed when checking with the database or the server.
<table>
<thead>
<tr>
<th>Date of attack</th>
<th>Description</th>
<th>Type of attack</th>
<th>Cause of attack</th>
</tr>
</thead>
<tbody>
<tr>
<td>11/28/12</td>
<td>Hacker steals 150k from school</td>
<td>Stolen Credentials</td>
<td>Weak application</td>
</tr>
<tr>
<td>11/25/12</td>
<td>Bank was asked to pay 42k to the hacker who has hacked the application on the cloud</td>
<td>Stolen Credentials</td>
<td>Weak application</td>
</tr>
<tr>
<td>10/26/12</td>
<td>Hacker steals 3.6 million social security numbers from the bank application on the cloud</td>
<td>Stolen Credentials</td>
<td>Weak application</td>
</tr>
<tr>
<td>10/15/12</td>
<td>Hacker hacked the Fairfac holiday site called Stayz and changed the bank details on the site</td>
<td>Stolen Credentials</td>
<td>Authentication is insufficient</td>
</tr>
<tr>
<td>09/18/12</td>
<td>Hacker steals 140k from lock poker account</td>
<td>Stolen Credentials</td>
<td>Authentication is insufficient</td>
</tr>
<tr>
<td>09/06/12</td>
<td>Thousands of the Guild War 2 accounts were hacked</td>
<td>Stolen Credentials</td>
<td>Authentication is insufficient</td>
</tr>
<tr>
<td>01/31/12</td>
<td>A Toyota employee was hacked and confidential information was stolen</td>
<td>Stolen Credentials</td>
<td>Authentication is insufficient</td>
</tr>
<tr>
<td>09/16/12</td>
<td>Nova Scotia website was attacked</td>
<td>Stolen Credentials</td>
<td>Authentication is insufficient</td>
</tr>
</tbody>
</table>

Table 1: Attacks on the clouds caused because of stolen credentials

All the attacks took place on the applications which are hosted on the cloud. In the thesis we
have worked on securing the credentials and authenticating in such a way that the plain text is not revealed.

1.3. **Our Contribution and Approach**

We have seen many outbursts and hacks on the user's credentials which include username and password. We have proposed architecture for a user to securely enter the username and password for authentication purposes. While checking the usernames and passwords for authentication, the check takes place on the encrypted data and not on the plaintext of username and password which shows that the authentication is strong and will lead to a strong application.

In this thesis, we have discussed various other ways of login authentication. We discussed several other existing methods to make the authentication happen in a secure way. Some of the methods discussed are Authentication using Reverse Turing Test, Lamport Scheme, and Authentication in cloud using One Time Password. We have discussed these models in detail in Chapter 2 and also discussed their contribution in authenticating the users and comparing them with the model we propose and seeing which performed better.

In our proposed architecture we have used the homomorphic encryption on the credentials. We have discussed Gentry's homomorphic encryption and homomorphic encryption with Goldwasser-Micali algorithm. We discussed the advantages and disadvantages of each of the algorithms and compared and listed the advantages of using the variant of Paillier's cryptography algorithm which we have used in this thesis. We have discussed the efficiency of the algorithms and the issues caused because of each of the algorithms. For instance, Gentry's homomorphic encryption generates 2 terabyte
of ciphertext for every bit of plaintext [25], and Goldwasser-Micali has problems such as time efficiency and length of the cipher texts [28]. Our contribution through this thesis is to show how feasible and how advantageous it is for homomorphic encryption to be applied on the data in the cloud.

We have developed a web application on the cloud which authenticates the user with the username and password; the information is stored in MySQL server and is encrypted. We have taken usernames and passwords of a variety of lengths. The web application server we have used is Google App Engine which happens to be an integrated server for Google Web Toolkit(GWT) for development. We have hosted the application, MySQL server and Tomcat application server on the cloud platform, Cloudfoundry.

The efficiency in regards to time taken to compute the data is calculated by using the model we proposed with the Paillier’s homomorphic encryption. The efficiency depends on two components in the scenario, length of the plain text and length of the key. From our results and observations, efficiency decreases as the length of the plain text increases, and efficiency decreases as the lengths of p and q increase. Due to the computation capacity of the cloud, the time taken for computation is still satisfactory for the large values of the key. We have shown that the proposed architecture and algorithm works well against the replay and man-in-the-middle attacks.

We have shown practically that the proposed architecture is safer in some circumstances by decreasing the password leaks. Credentials are protected because during authentication, the checking is done on cipher texts and not on the plain texts. The performance varies on various factors like length of the key and length of the plain text in our model. In the comparison we see that Authentication with the One Time Password model will fare better than our model in the case of user anonymity and mutual
1.4. Thesis Organization

Chapter 1 describes the problem and motivation and gives an overview of the contribution of this thesis to the problem. Chapter 2 discusses the related work where we have discussed various other models of secure login authentication of the user in the cloud. We also discuss the similarities and dissimilarities of these models with the model that we have proposed. We discuss which model fared better under given conditions and circumstances. We also discuss in chapter 2 about various encryption schemes which satisfy homomorphic property. In chapter 3, we discuss the proposed new model for secure authentication of the user. We discuss the Paillier's homomorphic encryption scheme that we use in our model. Chapter 4 discusses about the development environment, and the results we get after analysis. In chapter 5, we give our conclusions and also discuss about the future work to improve the model. The Appendix gives the procedure to activate cloudfoundry, to install MySQL and the Tomcat Application Server on the server.
Chapter 2

Background and Related Work

In this section we discuss some of the models which are already developed for making the login authentication secure. There are a number of methods for authenticating the user; some of them are password authentication, one time password method, smart cards and physiological behavior. In this section we discuss some notable schemes and discuss how similar and different they are with our proposed model. We also discuss encryption schemes which satisfy the homomorphic property. We discuss how similar and different they are with the encryption model we use in our model. We discuss if their performance is better than the Paillier’s algorithm under given conditions.

2.1. Secure Login Authentication Models

There are several methods of authentication. Password is the most common way of authenticating a user. There are other methods of authenticating a user such as smart cards, cryptography, but none of these have gained widespread acceptance as password based authentication.

2.1.1. Authentication with Reverse Turing Test

Reverse Turing Test (RTT)

Reverse Turing test is used to distinguish between a human and a machine. An example for RTT is Captcha, where there are distorted letters which can be identified by a human and cannot be identified by machine [31].

Most of the attacks on the password authentication occur because of the brute force method.
The intruder can launch the application and by brute force he can try a number of passwords until he gets a hit. Until recently, some methods have been followed to cut back the brute force hits, some of the methods are [31]:

- **Time expiring method**
  When the username and the password are entered into the account, the server will give a reply if they are valid credentials in more than 1 second and in this way, the intruder is not allowed to check many numbers of credentials in a given time.

- **Lock the Account**
  Give the user only limited number of times to enter the credentials. Sometimes the user may make a mistake in typing and for that reason a user should be given a chance for two or three times to enter the credentials. If the user exceeds the number of chances, then he gets locked out of the account.

These methods work well until the intruder becomes interested in breaking one particular account. Below are some of the scenarios when the above methods may never be triggered.

- The intruder may be trying to break several accounts at a time, so the time expiring method may not work here
- The intruder may be using different users one after the other so that no user is checked twice.

Getting hold of the usernames is much easier these days. In most of the companies, the email addresses are the usernames and also the valid bank account numbers are reasonably easy to generate. Part of the bank account number is the branch code, part is the hash code and part is the city code. Only a part of the number is random and the rest of it can be known if known some information. If the bank uses the account number as a username, then the account can be compromised without any disturbance.
Password authentication with the RTT [31] has been used for providing better resistance to the brute force attacks. RTT is the Reverse Turing Test, it helps to differentiate between a human user and a machine or the algorithm. The user has to satisfy both the username and password and also RTT to get access into the account.

The authentication of the user is done by following an algorithm [31]. The algorithm is shown in the Figure 3. The algorithm is as follows:

- The username and the password are requested.
- The username and password are entered.
- Login is determined if it is successful or not.
- If the login is successful, RTT is given to the user.
- User answers the RTT.
- If the RTT is correctly answered by the user, then the user is authenticated.
- IF the RTT is incorrectly answered by the user, then the user is invalid.
- If the login of the user fails, RTT is still given to the user.
- If the RTT is answered correctly by the user and the password authentication failed, he is denied access into the application.
- If the RTT is incorrectly answered by the user and the password authentication failed, he is denied access into the application.

RTT will mainly help in differentiating between the human user and an algorithm. In the devices like mobiles, RTT can take the form of sound to ensure that the human can understand it. Noise in the background should be added to ensure that the human and not a machine is actually working to get authenticated.
Figure 3: Password Authentication with RTT[31]
2.1.2. Login Authentication using cookies

We have been seeing that if the gmail account is being opened on a new machine for the first time, it asks for RTT after the username and password are entered. And if we are accessing our gmail accounts on a machine where we have accessed several times, it does not really ask for RTT, which means a cookie is being saved on our machine with the information of the user to know if the machine has been used for accessing the gmail account or not.

This is a smart way of authenticating the user. It first checks the machine for the cookie it saves which stores the user information. If the machine gives the information of the user, it goes easy on the user by not asking him for the RTT test. If the machine is new for the user, then the user has to go through the password authentication and the RTT test. If he fails the password authentication, even then he will have to go through the RTT test.
Figure 4: Login Authentication with Cookie [31]
The login authentication with the help of cookies is as follows:

- When the user enters the login credentials and is given the RTT, the server saves a cookie on the user’s machine which saves all the information of the user.
- When the user launches the application and enters the login credentials, the server will check if the cookie is present. If the cookie is present, then the regular machine of the user is being used and server will extract the identity of the user and the server will just ask the login credentials but not the RTT.
- When the user goes to the login page to enter the application, if the cookie is not found then the server thinks it is a new machine the user is using and will ask for the RTT along with the login credentials.

The advantage of this model is that, it takes the help of Reverse Turing Test to determine if the user is human or a machine. It authenticates the user smartly by evaluating when to check the user with RTT and when not to check the user depending on the cookie information it saves on the user’s machine. RTT is a great idea which asks a question which can be answered by a human and cannot be answered by a machine or algorithm.

2.1.3. Comparison between Authentication with RTT and the proposed model

In this model, as discussed there is a chance of impersonating the user. The intruder can start several instances and perform the brute force, which is not an impossible scenario. The intruder can also try to break in several users at a time. In this way, the trigger for locking the account on 3
failed logins will not go off. This is a security hole and when compared with the model we proposed, it is a big hole in security in the authentication with RTT. This model succeeds in distinguishing the problem if the entity that uses the machine is a human or a machine but some leaks are still possible.

2.2. Lamport Scheme

In order to impersonate the user, the intruder has to gain knowledge of the password of the user. The intruder can gain knowledge of the user’s password in three ways [34]:

- By reading the system’s password files, that is by gaining the access to the system.
- By reading the messages which are transmitted from the user to the server, that is by intercepting the messages in the transit.
- If the user himself chooses to disclose the credentials to the third party, then nothing can be done.

The Lamport Scheme [34] works on eliminating the first two ways of gaining the information about the user’s password. The scheme uses one way function, let the function be \( f(x) \).

A one-way function is defined as mapping of \( f \) from some set of words into itself, such that it satisfies the following conditions:

- Given the value of \( x \), value of \( f(x) \) can be calculated.
- If \( y = f(x) \), given the value of \( y \), it is not feasible to compute the value of \( x \).

Instead of storing the password on the system, the user will store the value of \( f(x) \) where \( x \) is the password. In this scheme, the user stores the sequence of computed passwords
on the user side. Let the series of passwords to authenticate the user be \(x_1, x_2, x_3, \ldots, x_i\).

Let the value of \(n\) be the counter, which is user-defined. It can be for example, 100 or 1000. Lamport has proposed that the \(i\)th password \(x_i\) will be equal to \(f^{n-i}(x)\) where \(n\) is the counter value and \(i < n\). So, when \(n = 1000\), the sequence of passwords to be sent from user to the server are \(f^{999}(x), f^{998}(x), \ldots, f(f(f(x))), f(f(x)), f(x)\) and the values on the server side to authenticate the user are \(x, f(x), f(f(x)), \ldots, f^{998}(x), f^{999}(x)\).

**Security offered by Lamport’s Scheme**

Suppose the intruder has read the password \(x_m\) sent from user to the server, next time for authentication, the password is not the same, it will be \(x_{m+1}\), so replay attacks are not possible. If the user communicates with several systems, then it follows the same method to authenticate with the other systems too.

### 2.2.1. Comparison of the Lamport model with the proposed model

In Lamport scheme, there is lot of overhead because of the computation for the password on the client side. A counter is present which has to be reset every time it strikes the maximum count. The Lamport scheme is the first of its kind which uses one time password [34]. One time password because, the password it sends from one time is not equal to the password it sends the next time. Every time the password has to be computed before sending to the server. So it takes up lot of computing time and overhead space.
2.3. Cloud Authentication model based on anonymous one time password

2FA (Two Factor Authentication) [33] is discussed in this section. The two factors are one time password (OTP) and Asymmetric Scalar Product.

A scheme has been proposed that has three main components: Data owner (DO), user set and the server(S). The work is divided into three parts: setup, registration and authentication. Setup and Registration are done only once while the authentication is done whenever the user wants to use the application.

UID and pwd are the username and password of a user and following are the steps to authenticate a user:

- DO calculates the keys by using the RSA signature, it selects two large prime numbers p and q, p is not equal to q, n = pq. □(p,q) = (p-1)(q-1). e and d are selected from integers such that, gcd(Ø(n),e) = 1, 1<e<Ø(n); d = e\(^{-1}\) mod Ø(n) = 1.

The public key and the private key are, (e,n) and (d,n) respectively.

- DO generates the saltkey which is a shared secret key. Data owner will generate the key called M and then A is computed.

\[ A = (pw_i^T \times M_i) \times (M_i^{-1} \times pw_i), \] and C = H(A||B),

where x refers to matrix multiplication and where || means concatenation and T means transpose function.

- DO finally sends (M\(^{-1}\),private key, saltkey) to the user and (UID,C,B,saltkey,public key) to the server. This completes the registration stage.

- Two factor authentication is explained here:

User performs the one time function on the password which is entered by the user.
By using the private key, he signs a new hash password

\[ \text{Sig} = H(K,\text{Saltkey})^d \mod n \]

- User sends the Sig, UID, K to the server and it is called the first factor
- Upon receiving the first factor, the server computes K’ and K”

\[ K' = \text{Sig}^e \mod n \]

And \( K'' = H(K,\text{Saltkey}) \)

And then \( \alpha \) is generated which is an integer.

If \( K' = K'' \), then S computes \( R = K'||\alpha \)

- User compares \( (H(K,\text{Saltkey})||\alpha) \) with R. If the result is true, user ensures from server authentication and then second factor is provided for authentication and the following term is calculated:

\[ Pw_q' = M^{-1} \times pw_q \times \alpha \]

- The second factor is obtained by the server then it updates C as \( C = H(C||K) \). E and C’ are computed.

\[ E = B \times Pw_q' = Pw_i^TMM^{-1}pwq, \]

\[ C' = H(H(E||K)||K), \text{ where } i=q \]

If \( C = C' \), then the user is authenticated.
2.3.1. Performance of Cloud Authentication Model

The performance of the model is shown below in graphical way:

![Graph showing performance of cloud authentication model](image)

Figure 5: Performance of cloud authentication model [33]

In the graph shown above, as the number of users who access the application increases, the time taken for computation increases. The time shown is not bad in the graph for the number of users. This is equivalent to 0.0256 seconds for a single user which is a good performance [33].

2.3.2. Comparison of the Cloud Authentication Model with the Proposed Model

This two factor authentication [33] is better than the one we proposed in this thesis in the case of time complexity, however, our algorithm has the advantage that it does not send the raw password to the server for authentication. The credentials are computed on the client side and the result is sent for
authentication. Cloud Authentication with One Time Password is better as it provides user anonymity, mutual authentication and also this algorithm achieves security with low cost and good performance and has no dependencies in the form of length of the key and the length of the message.

Authentication with one time password model is independent of the length of the key and the length of the message. It provides many more features like anonymity of the user, mutual authentication of the client and server; secure password change, freely chosen password, and session key agreement. The performance achieved with this algorithm is at the rate of 0.0256s for a user which is a very decent performance [33]. Our proposed algorithm deals with the homomorphic algorithm and gives a satisfactory performance as shown in the Results section.

2.4. Encryption

In the model we propose, we use the encryption algorithm which satisfies the homomorphic encryption. We discuss the types of encryption and see what the homomorphic encryption is. There are three types of encryptions; hashing, symmetric encryption and asymmetric encryption. Each of these encryptions has their own advantages, disadvantages and uses. Hashing is not as flexible as other encryption methods. All three types of encryptions rely on cryptography.

2.4.1. Basic Function

Encryption is used to change the readable and understandable information, called plain text into unreadable text called as ciphertext. There are a lot more advantages besides the confidentiality which it offers. These benefits can be realized by any of the following encryptions [35].
2.4.2. Hashing encryption

This encryption method uses hashes for encryption. Hashes are usually created by an algorithm or a hash function. A hash is usually very specific to the message involved, therefore if the message changes, the hash changes and gives clue if there is a potential attack.

A very important difference [35] between hashing and other types of encryptions is that, hashing is not reversible or decipherable, which means even if someone obtains the hash for deciphering the message, he will not be able to decrypt and learn the original message. Secure hashing algorithm is an example of hashing encryption.

2.4.3. Symmetric encryption

Symmetric encryption is called private key encryption. This is the oldest and the most reliable form of encryption [35]. Private key is used for the encryption and the term private key came into existence because the key should be secure and remains a secret for the encryption to take place. A sender encrypts because anyone with the private key can read the secure messages. A sender encrypts the message with the private key and receiver can read the decrypted messages by using the same private key.

This cipher is both stream cipher and block cipher. Whether the cipher is block or stream is decided by the amount of data encrypted. In a stream cipher, each character is sent and encrypted and each character encrypted is sent to the receiver, while a block cipher processes a block at a time. Some
of the symmetric algorithms are DES and AES. DES is the Data Encryption Standards, AES is the
Advanced Encryption Standards.

![Symmetric Encryption/Decryption process]

Figure 6 : Symmetric Encryption/Decryption process

2.4.4. Asymmetric Encryption

Asymmetric encryption is potentially more secure than symmetric encryption. This uses both
public and private key for the encryption and decryption of data. The use of two keys will dilute the
major weakness of securing the only key used in the symmetric encryption, since a single key need not
be used for the encryption [35].

In asymmetric encryption, the public key is used to encrypt the data to be sent. Private key is
with the receiver to decrypt the information. RSA and Diffie-Hellman are the two types of Asymmetric
encryption.
2.4.5. Homomorphic Encryption

This is a type of encryption that allows computations to take place on the cipher text to get the cipher text and it is the same result as the computations carried out on the plain text [8].

For instance, when a person adds two encrypted numbers and the other person can decrypt the result but cannot find out the individual values. The homomorphic encryption functions are malleable or very elastic in design perspective [8].

Usually the homomorphic function supports either addition or multiplication. If a particular homomorphic function supports both of them, then it is called fully homomorphic encryption. If this scheme is run, it is evaluated on the encryption of their inputs to produce an encryption of their outputs. As discussed before, this never decrypts the individual values; it can be run on the untrusted platforms, which implies that this is very helpful in the open platforms like the cloud computing.

The goal of homomorphic function is that the data is encrypted and is sent to the receiver
without decryption of the text. When the information is stored in the cloud as \( c(m) \), that is the cipher text of \( m \), calculations on the encrypted is done as \( f(c(m)) \), which means the calculations are done on \( c(m) \) and the information is not revealed by decrypting it on the cloud. In this algorithm the calculations are carried out on the cipher text.

The main idea of homomorphic function is not to reveal the individual values and they should remain encrypted and the computation is done on the encrypted values. After the computation is done, the result is computed and then it is decrypted, therefore the actual data is not revealed [8].

![Figure 8: Homomorphic Encryption [8]](image-url)
2.4.5.1. Fully Homomorphic and Partially homomorphic methods

Homomorphic over multiplication

Let $f$ be a function. Let $a$ and $b$ be two values. The function $f$ is said to be homomorphic over multiplication if it satisfies the following property:

$$f(a \times b) = f(a) \times f(b)$$

For instance, in our case, domain is bigintegers and range is bigintegers.
The function $f$ of product of $a$ and $b$ should be equal to product of $f(a)$ and $f(b)$.

Homomorphic over addition

Let $f$ be a function. Let $a$ and $b$ be two values. Here, $f$ is said to be homomorphic over addition if it satisfies the following property:

$$f(a + b) = f(a) + f(b)$$

The function $f$ of the sum of $a$ and $b$ should be equal to sum of $f(a)$ and $f(b)$.

Fully Homomorphic Method

A function is said to be fully homomorphic if it satisfies both homomorphic property over addition and multiplication.
**Partially Homomorphic Method**

A function is said to be partially homomorphic if it satisfies either homomorphic property over addition or multiplication.

**2.4.5.2. Advantages of Homomorphic Encryption**

With the help of homomorphism, all the sensitive information is encrypted, therefore it is not of much use to the bad guys.

Let us look at some encryption schemes which satisfy homomorphic property. Some of the algorithms we take a look in this section are Gentry’s algorithm and Goldwasser-Micali algorithm. We discuss the advantages and disadvantages of these encryption schemes and the time complexity of the encryption schemes and compare the algorithms with the Paillier’s encryption scheme.

**2.4.6. Homomorphic Encryption Schemes**

**2.4.6.1. RSA Algorithm**

The algorithms discussed in this section use the basis of RSA. The RSA algorithm is discussed as follows [36]:

- Choose two random prime numbers p and q.
- The product of p and q is computed and denoted by n.
- $\emptyset(n) = (p-1)(q-1)$ is calculated.
- Choose $e$ such that $1 < e < \emptyset(n)$ and $\gcd(e, \emptyset(n)) = 1$.
- Determine the value of $d$ such that $e.d = 1 \mod \emptyset(n)$.
- Public key is $(n, e)$ and private key is $(n, d)$.

**Encryption**

Let $m$ be the message to be encrypted and $c$ is cipher text.

Then, $c = m^e \mod n$

**Decryption**

$c$ is the ciphertext. If $m$ has to be decrypted with the private key $(d, n)$

then, $m = c^d \mod n$

Let us look how the multiplicative homomorphism is obtained in RSA algorithm. It is shown as follows. Decrypting the product of two cipher texts will result in the product of the plaintexts.

\[
\square_{\text{RSA}}(x) \times \square_{\text{RSA}}(y) = x^e y^e \mod n
\]

\[
D(\square_{\text{RSA}}(x) \times \square_{\text{RSA}}(y)) = (x^e y^e)^d \mod n
\]

\[
= x^{de} y^{de} \mod n
\]

\[
= xy \mod n
\]

Here, $\square_{\text{RSA}}$ is the RSA encryption function.
2.4.6.2. Gentry's Algorithm

Gentry's algorithm is the first of its kind algorithm in homomorphic encryption. The algorithm is discussed below [25][38]:

The Gentry's algorithm has three parts, key generation, encryption and decryption and is defined as follows:

**Key Generation:** It is denoted by \( p \), and is randomly generated \( \lambda^2 \) bit odd integer

**Encryption:** If \( m \) is the plaintext, to encrypt \( m \), where \( m \in \{0,1\} \). The ciphertext \( c \) has to be picked which will have same parity as \( m \) and \( c = pq+m' \), \( p \) is the key and \( q \) is random which is \( \lambda^5 \) bit integer.

**Decryption:** The decryption of the ciphertext is \( m = (c \mod p) \mod 2 \).

Homomorphic property can be shown by multiplication and addition of two ciphertexts [25].

**Multiplication**

\[
\Box(x) \times \Box(y) = (m_1 + pq_1)(m_2 + pq_2)
\]

\[
D(\Box(x) \times \Box(y)) = (m_1 m_2 + m_1pq_2 + m_2pq_1 + p_1q_1q_2 \mod p) \mod 2
\]

\[
= m_1 m_2 \mod 2
\]

**Addition**

\[
\Box(x) + \Box(y) = m_1 + pq_1 + m_2 + pq_2
\]

\[
D(\Box(x) + \Box(y)) = (m_1 + m_2 + pq_1 + pq_2) \mod 2
\]

\[
= m_1 + m_2 \mod 2
\]
There are a number of challenges in applying Gentry's algorithm in security for the cloud. To obtain \( O(2^n) \) is approximately, \( 2^{80} \) which is close to 2 terabyte of ciphertext for a single bit of plaintext is used [25].

### 2.4.6.3. Goldwasser-Micali Algorithm

This is another homomorphic encryption and decryption algorithm which satisfies the homomorphic property. It is discussed below[28]:

There are three main components in the algorithm Key Generation Algorithm, Encryption Algorithm and Decryption Algorithm.

#### Key Generation Algorithm

- The key generation algorithm generates \( p \) and \( q \) which are two large prime numbers.
- \( n \) is calculated by multiplying \( p \) and \( q \) and a non-residue number \( x \) is calculated for which the Jacobi number is 1.
- The public key is \( \{x, n\} \) and the private key is \( \{p, q\} \).

#### Encryption Algorithm

- Let \( E \) be the encryption algorithm, it takes in the message \( m \) belongs to \( \{0, 1\} \) and it takes the public key \( \{x, n\} \) as input and the output is the ciphertext, \( c = (y^2)(x^m) \text{mod } n \) and \( y \) is a randomly chosen integer.
**Decryption Algorithm**

- Let the decryption algorithm be $D$, it takes the input as ciphertext $c$ and the secret key $\{p,q\}$
- It gives the output $m$, $m = 0$ if $c$ is quadratic residue
  
  $m = 1$ if otherwise.

It is known that the Quadratic Residue problem is unsolvable for the time being, if it breaks, this algorithm will be at a risk [28]. It is also known that this algorithm encrypts the data bit by bit. With this property, the security is very high, but the main question with efficiency props up. It takes lot of time to encrypt the data bit by bit and also the length of the ciphertext generated is several hundred times the size of the plain text.
Chapter 3
Methodology

3.1. Paillier’s algorithm

Paillier's [18][26] homomorphic encryption’s description is as follows:
This is used for the encryption:

\[ E(m) = g^m r^n \mod n^2 \]

Where \( E(m) \) is the encrypted data
\( m \) is the data which has to be encrypted
\( g \) is a random biginteger
\( r \) is a random biginteger
\( p \) and \( q \) are random prime numbers
\( n \) is the product of \( p \) and \( q \).

The decrypted homomorphic function is used for the decryption purpose and it is as follow:

\[ M = L(E(m)^{\lambda} \mod n^2). \mu \mod n \]

\( L \) is defined as,
\( L(\mu) = (\mu-1)/n \)
Where \( \mu = (L(g^{\lambda} \mod n^2))^{-1} \mod n \)
\( E(m) \) is the encrypted value, waiting to be decrypted.
3.2. Cloud Platforms

The type of the cloud platform makes a lot of difference. There are different cloud vendors available in the market as discussed and we need cloud from a member that satisfies our requirements. We considered using one among Amazon Cloud or Microsoft or cloudfoundry, which is Platform as a Service (PaaS). We have implemented using cloudfoundry.

3.3. Proposed Architecture

![Common architecture for cloud services](image)

Figure 9: Common architecture for cloud services

The usual cloud architecture is shown in Figure 10. When the cloud user wants to access an application on the cloud, it gets authenticated through the third party.

We propose a model to make the process in between the user and the server secure. We propose the way in which the user gets authenticated without getting its credentials revealed.

Let us consider an application which requires login and the credentials should be present in the
cloud [18]. We have developed a web application which authenticates the user on the cloud. The credentials are computed and stored in database.

The process in the model we propose is discussed below:

Figure 10: The proposed architecture

Figure 10 shows the proposed architecture,

- The user sends the computed values of username and password in the encrypted format and are stored in the database during the registration
- The data sent by the user is checked against the database for the valid credentials
- For the checking with the database, the homomorphic property is used
- When they satisfy the homomorphic property, nonce is given to the user for checking against the replay attacks
User gives some information involving the computation with nonce.

Computed values are sent to the server and it is checked if the value received by server is replayed information or original information.

The above mentioned points are the flow of the control that takes place. The security leak will be negligible because the data is not being exposed anywhere.

### 3.4. Proposed Algorithm

The two phases in the algorithm are registration and authentication.

We use the following notation:

- U and P are the username and password, E is the Paillier's Encryption function, D is the Paillier’s Decryption function.

#### Registration:

E(U) and E(U+P) are sent from user to the server and stored in the database. \( K_r \) is the key for decrypting the information in the database.

#### Authentication:

\( U \rightarrow S: E(U); [E(U) \times E(P)] \mod n^2; K_a; K_r \) to server; \( K_a \) is the key for decrypting information for authentication.

**Server:** \( D(E(U+P), K_r) = D(E(U)*E(P),K_a) \) and \( (E(U), K_r) = (E(U),K_a) \)

If both the conditions fail, then the authentication fails.

If the conditions are met, then it is checked if the credentials are replayed or original.

Two large prime numbers \( p_1 \) and \( q_1 \) are generated and \( n_{\text{replay}} = p_1 \times q_1 \)

\( e \) and \( d \) are selected such that \( e.d \mod n = 1 \) where \( e \) is a prime number and \( d \) is an integer.

\( e \) is on the server side for for further calculations to check against the replay attacks.

\( S \rightarrow U: N_s = N_{s_c} \text{ xor } E(U) \text{ xor } E(U+P); d; n_{\text{replay}} \)
**User:** \( Ns_{\text{client}} = Ns \oplus E(U) \oplus E(U+P) \)

\( rc \) is a random large number generated on the client side

\( X_1 = g^{rc \cdot Pw} \mod n \)

\( X_2 = S \cdot h^{rc \cdot Ns_{\text{client}}} \mod n \)

\( H = g^{Pw \cdot d} \mod n \)

\( S = U^d \mod n \)

\( U \rightarrow S: X_1; X_2 \)

**Server:** If \( X_2^e = E(U) \cdot X_1^{N_s} \mod n \)

If the condition is satisfied, then there is no replay attack.

If the condition is not satisfied, then there is a replay attack.

### 3.4.1. Security offered by the proposed algorithm and architecture

**Replay Attacks**

Replay attack is a type of network attack where the information sent by the user is replayed by the intruder, that is, the message sent is intercepted and resent by the intruder.

We see that \( E(U) \) and \( E(U) \cdot E(P) \mod n^2 \) and \( K_a \) and \( K_r \) are sent from user to the server. Let us assume that there is a replay attack and these values are saved. \( X_1 \) and \( X_2 \) are sent from user to the server and they involve the value of nonce. Let us assume that these values have been replayed too. The authentication will not be successful because the nonce which is sent by the server is not being used in the calculation and to make it further random, the user will include a random number called \( rc \) in calculating the value of \( X_1 \) and \( X_2 \) to send to the server.
The pair (e,d) are calculated on the server side and e remains at the server. For every authentication session, the value of d, e and nonce are unique. The combination of all these values is always unique. For the computation of $X_1$, it needs the value of password is not sent to the server before, even if all the values are known to the intruder guessing the password is not easy and the replay attacks are not possible.

**Man in the Middle Attacks**

In the Man-in-the-Middle attack, the intruder eavesdrops and later replays the information which is sent. Man-in-the-Middle attacks are successful when the intruder successfully makes both the parties believe that he is the genuine other party, he has to make the user believe that he is the genuine server and server should believe that he is the genuine user. It is not possible because the computation of $X_1$ will include the value of password.
The flowchart for model is:

start

Enter the username and password for registration

encrypt credentials using homomorphic function

Store in database

Enter the username and password for authentication

Encrypt the credentials

homomorphic property and if the information is replayed

yes

Forward the user to the application

no

Figure 11: Flowchart for proposed model
Chapter 4
Experiments and Results

Figure 12: The architecture of the system

The architecture of the system is shown in the figure 12.

Let us walk through the actual experiment which is carried out. We have deployed a web application for authenticating the user on the cloud. The database has the credentials stored in encrypted format. When the user enters the username and password, it is checked against the encrypted data and the credentials are not decrypted while checking. When the credentials do not match, the user is not authenticated, if they match it is checked if the information sent by the user is replay information or original. The checking of the credentials is done on the server. The flow of the program is as follows, here we will use the notation used in section 3.3 and 3.4:

- All the usernames and passwords are stored in the database in encrypted format.
- The client side reads the username and password and computed information and is sent to the
server as $E(U)$ and $E(U+P)$ during the registration of the user.

- The client sends $E(U)$, $E(U^*)E(P)$, $K_a$, $K_r$ to the server to check if the user is valid or not
- The server calculates the result of the request and if the condition is not met the request is rejected
- If the condition is met, the server sends some information to the server and asks for some information from the user to check if the information is replayed or original
- The client will send the values $X_1$, $X_2$ to the server
- The server will check the condition to see if the information is original or replayed
- If the information is not replayed, i.e. if the condition is met, the client is given access to the application, if the condition is not satisfied, the client is not given access to the application

The web application is hosted on the cloud. Tomcat application server and MySQL are installed on the cloud. We have used cloudfoundry, which is a Platform as a Service (PaaS).

### 4.1. Efficiency of the proposed architecture

#### 4.1.1. Efficiency when the length of the plain texts varies

The size of the plaintext is tested against the time taken in cloud for computing to see the advantage of encryption in the cloud over the Gentry's homomorphic encryption.

<table>
<thead>
<tr>
<th>Length of plaintext (in digits)</th>
<th>Time taken for computation on data (in milliseconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>3737</td>
</tr>
<tr>
<td>48</td>
<td>3911</td>
</tr>
<tr>
<td>160</td>
<td>4268</td>
</tr>
<tr>
<td>320</td>
<td>4922</td>
</tr>
<tr>
<td>640</td>
<td>6098</td>
</tr>
<tr>
<td>1280</td>
<td>8967</td>
</tr>
</tbody>
</table>

Table 2: Time taken for computation according to length of the plaintext
These values in the table are taken for $p$ and $q$ which are 64 bit long and everything is kept constant. The length of the username and the password are the same. In the graph shown above, the x-axis is the length of plain text denoted in the number of digits, and y-axis denotes the time taken for computation in milliseconds. It is shown that, as the length of the plaintext increases, the time taken to compute the data increases gradually.

In the above table, as the length of the plaintext is increasing, time taken for computation on the ciphertext which is encrypted using the algorithm is increasing. From the observation, the time taken is not increasing exponentially, which is a positive sign for applying such technology in practical purposes.

This is definitely far better than Gentry's homomorphic function because, for a bit of data, 2 terabyte of ciphertext is generated which makes it not workable in practical purposes.
In Gentry's algorithm of homomorphic encryption, the terabyte of data would take hours or days to compute, though huge amounts in terms of gigabytes of data cannot be worked upon using the homomorphic function for practical purposes.

### 4.1.2. Efficiency when the length of the $p$ and $q$ vary

The size of the pair $p$ and $q$ are also tested against the time taken in cloud for computing. Let us see the time taken for computation when the $p$ and $q$ pair size changes with the given fixed values of the plaintext. In this we have taken the length of the plaintext fixed to be 10 digits.

<table>
<thead>
<tr>
<th>Size of $p$ (number of bits)</th>
<th>Size of $q$ (number of bits)</th>
<th>Time taken for computation (milliseconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>40</td>
<td>40</td>
<td>4019</td>
</tr>
<tr>
<td>64</td>
<td>64</td>
<td>4365</td>
</tr>
<tr>
<td>128</td>
<td>128</td>
<td>5023</td>
</tr>
<tr>
<td>256</td>
<td>256</td>
<td>5564</td>
</tr>
<tr>
<td>512</td>
<td>512</td>
<td>6035</td>
</tr>
<tr>
<td>1024</td>
<td>1024</td>
<td>6785</td>
</tr>
<tr>
<td>2048</td>
<td>2048</td>
<td>7987</td>
</tr>
</tbody>
</table>

Table 3: Time taken for computation according to length of $p$ and $q$
Figure 14: Efficiency when length of the key varies, x-axis is length of p and q; y-axis is the time taken for computation in milliseconds; length of p and q is in number of bits

In the above table, it is shown that the time taken for computation of the data according to the length of the public and private keys. As the length of the keys increase, the time taken for computation on the data is increasing. We have implemented using the length of p and q to be 512 bits, which gives good security. As the length increases to 2048 bits and beyond, it may not be used for industrial purposes.
4.1.3. Length of the Plain text vs Length of the Cipher text

<table>
<thead>
<tr>
<th>Length of plaintext (number of digits)</th>
<th>Length of ciphertext (number of digits)</th>
</tr>
</thead>
<tbody>
<tr>
<td>135</td>
<td>1020</td>
</tr>
<tr>
<td>314</td>
<td>1021</td>
</tr>
<tr>
<td>537</td>
<td>1022</td>
</tr>
<tr>
<td>882</td>
<td>1019</td>
</tr>
<tr>
<td>1231</td>
<td>1021</td>
</tr>
</tbody>
</table>

Table 4: Length of plaintext vs length of cipher text

Figure 15: Length of plain text vs length of cipher text
Gentry’s algorithm generates 2 terabyte of ciphertext for every one bit of plain text [25]. In Goldwasser-Micalo algorithm, the ciphertext is several hundred times the size of the plain text [28]. From the Table 4, it shows the length of the ciphertext is constant with the length of the plain texts varying. This can be graphically illustrated in Figure 15.

4.2. Comparison of Paillier's with Gentry's and Goldwasser-Micali algorithm

The comparison among these algorithms can be on the basis of efficiency, length of the ciphertext, time taken for the computation on the data. Let us look at all these criteria

4.2.1. Time taken for computation

The whole idea of this research is to see if the models are good in terms of time taken for computation or efficiency in time, Paillier's as we saw in the results, it is moderate for plaintext of size around 1000 digits, for Gentry's algorithm the time for computation is very bad because it gives out 2 terabyte of ciphertext for every bit of plaintext [25], which takes huge amount of time to compute, and for Goldwasser-Micali algorithm [28], this encryption takes bit by bit, so the time complexity is huge for this algorithm too. When compared, Paillier's algorithm is better than the other two algorithms and give out results in better time.

4.2.2. Length of the ciphertext

The length of ciphertext plays a great role in comparison among the algorithms. Paillier's gives out medium length of the ciphertexts out, not too long and not too short at the same time. We have seen
that Gentry's algorithm gives out very huge length of ciphertext, it is said to give out 2 terabyte of ciphertext for every 1 bit if plaintext [25]. When Goldwasser-Micali algorithm is taken into picture, the encryption is done bit by bit, so it will be significant amount of ciphertext generated [28]. In this way too, Paillier's algorithm is proved to be better than the other two.

4.3. Comparison of authentication models with the proposed model

▲ The proposed model works well with the password leaks unlike the authentication with RTT.
▲ The authentication model with RTT distinguishes well between the human and a machine, but in the proposed model, we do not yet have functionality to differentiate if the brute force attack is taking place or not.
▲ The Lamport model is the first of its kind of one time password, the password sent for authentication by the user to the server is different every time, hence reducing the chances for replay attacks.
▲ Disadvantage of Lamport scheme is that the counter has to be reset every time it reached the zero and there is computational overhead to compute the password on the user side.
▲ Amongst all the authentication models, 2FA model scores high with excellent performance. The performance of the proposed method is good until the length of the plaintext and the key are of moderate length. 2FA model does not have any dependencies.
▲ 2FA model has a setup to win against man in the middle attacks, denial of service while the proposed model wins against man in the middle attacks and replay attacks but cannot win against denial of service. If the intruder can get into the system with the brute force method, then the intruder can deny the service to the user.
▲ 2FA model has several advantages like it can offer user anonymity, freely chosen password,
secure password change, mutual authentication, and secure session management.

The proposed model is better than the authentication with RTT, Lamport is the first of its kind in using the one time password approach, and in comparison to 2FA, 2FA scores high because of its performance and various features it offers.
Chapter 5

Conclusion and Future Work

In this section we summarize the work done in this thesis. We discuss the future scope of the model along with the other future work which can be done in this field.

5.1. Conclusion

We understand that homomorphic functions have never been used so far in practical applications. The possible reason is that the method is not efficient in terms of time taken for computation and length of the cipher text generated using Gentry's algorithm.

The algorithm and the system architecture which are discussed in the thesis are applied in the cloud environment and it is shown that the time taken for moderate size of data is feasible in industry level, the time for computation increases as the size of the plaintext increases and as the lengths of the p and q increase.

The applications on which the algorithm and the data have been tested are Tomcat Application Server, MySQL server and Java programming language. The proposed algorithm is built on RSA, and RSA is secure because of the fact that there exists no algorithm for factorizing the integers into the
prime factors in polynomial time. Not only RSA, but also the internet security will be at stake if the Riemann hypothesis is solved [37]. Many attempts have been made like Fermat's algorithm that offers the primality test to determine if the number is a prime number or not.

5.1.1. Summary of Results

- Homomorphic Encryptions lead to greater security because, the plaintext is not revealed during the computation on the data.
- For the plaintext of moderate lengths, the efficiency in terms of time taken for computation is less and can be used in industry for limited uses.
- The use of Paillier’s algorithm along with the architecture provides better use in terms of security.
- The efficiency is dependent on size of the keys and keys of small length are not good for security.
- The proposed model is better than the model called authentication with RTT.
- The proposed model will score better than the Lamport scheme because of the computation overhead to calculate the password every time and the counter has to be reset every time it reaches zero.
- Two factor authentication model fares better than the proposed model in terms of performance. For a single user the performance is about 0.0256ms which is better than the proposed model.
5.2. Future Work

To optimize the homomorphic encryption and to apply in fields in the industry which will make a major difference in technology and to the people. Efficiency can be increased further and bigger amounts of data can be dealt. The homomorphic property can be applied on search too as an extension of authentication. A fully homomorphic encryption algorithm which may not have its base on RSA will be a huge leap in the cloud computing.

Some of the future work on the model is listed below:

- There should be flexibility in the model for incorporating changes in the future. If the authenticating rules or policies have to be changes, then there should be a way to incorporate them.
- The proposed model cannot win against denial of the service attacks, which is a very serious problem.
- The flexibility should be given to the user for password change and user anonymity, this is where 2FA scores better.


[4] “Cloud computing is picking up traction with businesses, but before you jump into the cloud, you should know the unique security risks it entails”, Internet: http://www.infoworld.com/d/security-central/gartner-seven-cloud-computing-security-risks-853


[10] “Microsoft vs. Amazon Clouds: 5 Key Differences”, Internet: http://www.cio.com/article/632213/Microsoft_vs._Amazon_Clouds_5_Key_Differences


[37] “Maths Holy Grails could bring disaster to Internet,” Internet: http://www.guardian.co.uk/technology/2004/sep/07/highereducation.science

Appendix

Developing Application on cloudfoundry

The application has to be deployed in the cloud by using the following steps as shown below:

**Define a New Server:**
- Window > Show View > Servers
- Click new server wizard or right click and choose New > Server
- Select Vmware folder and select Cloud Foundry
Define a New Server
Choose the type of server to create

Select the server type:

- Apache
- Basic
- IBM
- JBoss
- ObjectWeb
- Oracle
- VMware
- Cloud Foundry

Publishes and runs J2EE Web projects to Cloud Foundry.

Server’s host name: localhost
Server name: VMware Cloud Foundry

Figure: Defining New Server
Specify a display name for the instance of the server in the cloud. Servername should be localhost. Click on Next.

Figure: Setting Cloud Foundry Account

- The target should be selected from the URL list, the choices are, Vmware Cloud Foundry, Local Cloud, Microcloud
- If microcloud is selected, the domain registered for the microcloud is entered.
The email address and password should be entered for the cloud foundry target, the email which should be entered should be pre-registered at website. Validate the account and click on finish.

**Defining Application Details:**

- In Eclipse, Windows > Show Views > Servers to display the list of servers.
- To deploy the applications drag into the servers view
Binding the services to the application has to be done before the deployment or we can unselect the Start Application on Deployment and then we can bind the services to the application.

Once deployed, the applications can be seen in the servers view and on double click the editor will open up and all the options to start, stop, restart, and update the application can be seen.
Define Application Services:

In order to bind the application to the services, the service has to be defined. Following are the steps to define a service:

- In the servers view, click on the application.
In the services section, click on the Add Service.
Give the name and type of the service.

Bind Application and Services:

When the application and the servers are binded, the cloudfoundry integration extension will update the configuration files to define the servers and during binding the application should not be running.
Figure: Binding Application and Service