CHAPTER 4
AUTHENTICATION SCHEMES TO PREVENT SQL INJECTION ATTACK USING CRYPTOSYSTEM

This section proposes three authentication schemes to prevent SQL Injection Attack (SQLIA) using cryptosystem. The first scheme uses ElGamal cryptosystem for authenticated query encryption in user validation and this provides high security in web applications. The second scheme is developed by using AES (Advanced Encryption Standard) to encrypt the username and password by using this data login query is generated. Server decrypts the username and password and compare with user account database. The third scheme employs hybrid encryption method by combining (AES and RSA) to prevent SQL Injection Attack from unauthorized accessing of the web application. The following section describes the proposed authentication schemes.

4.1 PREVENTION OF SQL INJECTION ATTACK USING CRYPTOSYSTEM

SQL Injection Attack (SQLIA) targets web applications that employ database services. These applications accept the user inputs to generate SQL statements during runtime. In SQLIA an attacker may generate false SQL queries by using forged username and password. The attacker could manipulate the database records by using SQL Injection Attack. An attacker use SQL Injection Vulnerability as a rudimentary IP/Port scanner of the internal corporate network.

In the existing architecture, user submits the username and password to the web server through web browser. The query is created by using query interpreted along with service request. The requested query is sent to the database server. Finally, the response from the database server is sent to the web browser as shown in Figure 4.1.
In the following architecture Figure 4.2 shows that the user submits the username, password and user encryption key. The query will be generated by the application server and encrypt it by using user input key. The encrypted query is sent to database server. Database server decrypts the query and verifies the username and password from the user account table maintained in database. If the user is a valid one then it allows them to access the record, otherwise it rejects the query.

This thesis proposes three query encryption schemes:

1) Authenticated Query Encryption using ElGamal
2) Authentication mechanism for preventing SQL Injection Attack using AES
3) Authentication scheme using Hybrid Encryption
4.2 AUTHENTICATED QUERY ENCRYPTION SCHEME FOR PREVENTING SQL INJECTION ATTACK (AQE–PSQLIA)

Web application uses public–key cryptography for encryption and digital signature for authentication. This section proposes an authentication method (AQE–PSQLIA) for preventing SQL Injection Attack. The proposed authentication method is developed by using ElGamal cryptosystem. In this method every user must have a server public key. The proposed method consists of three phases, 1) Registration Phase 2) Login Phase 3) Verification Phase.

In the proposed scheme, characters in SQL query (plain text) are first converted as tokens and each token is encrypted by using ElGamal. The encrypted tokens are organized as a final encrypted query. Simple numerical example and coding module for each phase is given below.
The proposed scheme has two important contributions

1. SQL Injection Attacks can occur through injecting additional SQL statements in user input. This can be prevented in the proposed technique by applying encryption algorithms in query level.
2. The proposed scheme needs minimum computational cost and it is negligible.

**ElGamal Cryptosystem**

The proposed scheme is developed by using ElGamal cryptosystem. This section illustrates the basic information about ElGamal cryptosystem. ElGamal public–key cryptosystem is proposed by ElGamal in 1985. The ElGamal scheme can also be used to create both digital signature and encryption schemes. The security of ElGamal is based on the difficulty of calculating discrete logarithms in a finite field. In ElGamal’s digital signature scheme, each user has a key pair, which includes a private key and a public key. Select a prime \( p \), a generator element \( g \) and a random number \( x \), such that the two elements \( (x, y) \) are smaller than \( p \). Here \( x \) denotes user’s private key and \( \{ y, g, p \} \) denote user’s public key, where \( y = g^x \mod p \). When Alice wants to sign a message \( M \) and send its digital
signature to Bob. Bob can verify whether the signed message was really signed by Alice. The process is as follows.

**ElGamal Protocol**

Public key: \((g, Y, p)\) where \(p\) is a prime, \(g\) is a primitive element of \(Z_p\), \(X\) is an integer \(1 \leq X \leq p - 1\), and \(Y\) is an integer of \(Y = g^X \mod p\)

Private key: The integer \(X\)

**Initial setup:**

Alice obtains Bob’s public key \((g, Y, p)\)

For each message \(M\) \(Alice \rightarrow Bob:\)

Step 1: Alice selects a random number \(k\), here \(1 \leq k \leq p - 1\)

Step 2: Compute \(r = g^k \mod p\) and \(s = My^k \mod p\)

Step 3: Alice sends the cipher text message \(c = (r, s)\) to Bob

Step 4: Bob deciphers the cipher text message and recovers the original message as \(M = r^{-X} s \mod p\)

The correctness of the deciphering is verified as follows:

\[ r^{-X} s = (g^k)^{-X} My^{kX} = Mg^{X^2} = M \]

**Security of ElGamal Cryptosystem**

a) An adversary might attempt to forge A’s signature on \(M\) by selecting a random integer \(k\) and computes \(r = g^k \mod p\) and then the adversary must then determines \(M = x r + k s \mod (p - 1)\). If the discrete logarithm problem is computationally infeasible, the adversary can do no better than to choose an \(s\) at random; the success probability is only \(\frac{1}{p}\), which is negligible for large \(p\).

b) A different \(k\) must be selected for each message signed; otherwise, the private key can be determined with high probability. Once \(k\) is known, \(g\) is easily found.
4.2.1 Proposed AQE–PSQLIA Scheme

The proposed scheme consists of three phases: Registration Phase, Login Phase and Verification Phase. Every user has to register as a new user to the server with unique username (User Id) and password. The proposed architecture model is given in the following Figure 4.3 illustrates the registration phase and Figure 4.4 illustrates the login and verification phase (detection and prevention phase).

![Figure 4.3: Registration Phase in AQE–PSQLIA](image1)

![Figure 4.4: Login and Verification Phase in AQE–PSQLIA](image2)
The following parameters are used in this scheme:

**Basic parameters**

- $p$: a large prime number
- $ID_i$: User Id
- $PW_i$: Password
- $X_S$: private key of Server
- $Y_S$: public key of Server ($Y_S = g^{X_S} \mod p$)
- $g$: primitive root value from $Z_p$

**Registration Phase in AQE–PSQLIA**

The new user registers with the server along with userId and password through registration phase. The following steps are performed during registration phase.

**Registration Phase**

**Begin**

1. Step 1: Select User id $ID_i$ and Password $PW_i$
2. Step 2: Send a registration request along with ($ID_i$, $PW_i$) to server.
3. Step 3: Server receives the request and adds it as a new user and server sends a registration conformation
4. Step 4: User has to maintain $ID_i$, $PW_i$ and $Y_S$. Server maintains $ID_i$ and $PW_i$ for every user

**End**

**Login Phase in AQE–PSQLIA**

In this phase, users enter into the web server for accessing the sensitive data from database server. This phase has a wild card detection module. This module is used to detect the suspicious character from the user input. The following steps are performed in login phase.

89
Login Phase

Begin

Step 1: User enters the Username (User Id )ID_i and Password PW_i
Step 2: Check suspicious (meta) characters from user inputs
Step 3: Generate a SQL query for user login (Query_result)
Step 4: Encrypt the Login SQL query by using public key of server

\[ E_{i} (Query\_result) \] and send this encrypted login SQL query to Server

End

Verification Phase in AQE–PSQLIA

This phase is involved in application server itself. In this phase, user validation is performed through the verification of username and password from user account table. The following steps are involved in verification phase.

Verification Phase

Begin

Step 1: Server receives the encrypted login SQL query \[ E_{i} (Query\_result) \]
Step 2: Decrypt this query by using servers private key \[ D_{i} (E_{i} (Query\_result)) \]
Step 3: Server checks the username and password validity from the decrypted SQL query. If it is valid then it accepts the user login request and provides access rights. Otherwise, the login request is rejected

End

4.2.2 Coding Model for Proposed AQE–PSQLIA

This section provides coding model for three phases involved in the proposed scheme. The coding part of the proposed scheme is developed using core JAVA. The following section provides the coding part of Wild-Card Checker, SQL Query Generator, Authenticated Query Encryption, Query Decryption and User validation.
**Wild-Card Checker Module**

In this module, the user input is validated by using wild-card checker, which is statically generated in application level. The user data is validated by applying suspicious character detector. The wild-card checker module coding is given below (Figure 4.5).

```java
Public static int staticquerychecker(String Query_result)
{
    String Query_result = Query_result;
    if (Query_result.indexOf("drop")>0 || Query_result.indexOf("tracat")>0) ||
        Query_result.indexOf("")>0 || Query_result.indexOf("--")>0 ||
        Query_result.indexOf("@")>0 || Query_result.indexOf(",")>0)
    {
        return(1) // Query validation fails
    }
    return(0) //Query validation true
}
```

**Figure 4.5: Wild-Card Module in AQE–PSQLIA**

**SQL Query Generator Module**

```java
//... Read Input from user
System.out.println("Enter User Name:");
name = in1.nextLine();  // Read one line from the console.
System.out.println("Enter Password:");
Password = in1.nextLine();
in.close();              //Note 2  //... Display Query Result
String Query_result="SELECT * FROM user_account WHERE
user_name="+name+" AND Password=" +password;
```

**Figure 4.6: SQL Query Generator Module in AQE–PSQLIA**
The user input is combined with the static code in the application and a dynamic SQL query gets generated. In the query generator module, username and the password are received from the login page and corresponding SQL query will be generated. The SQL query generator coding is given in Figure 4.6.

**Authenticated Query Encryption Module**

```java
String Encryptor( String *Query_result, int p, int y, int g)
{
    //… Characters are separated from query and apply encryption
    for (int j=0;j<Query_result.length();j++)
    {
        r=1; s=Query_result[j];
        for(int i=1;i<=k;i++)
        {
            r=(r*g)%p;
        }
        for(int i=1;i<=k;i++)
        {
            s=(s*y)%p;
        }
    }
    Return (String EQuery_result);
}
```

**Figure 4.7: Query Encryption Module in AQE–PSQLIA**

In the query encryption module, each character is read from the SQL query and applies encryption by using ElGamal cryptosystem. The sample coding for query encryption module is given in Figure 4.7.
Query Decryption Module

```java
String Decryptor( String *EQuery_result, int x, int g) {

    //… Characters are separated from encrypted query and apply decryption
    for (int j=0;j<EQuery_result.length();j++)
    {
        int a=(p-x-1);
        m=1;
        for(int i=1;i<a;i++)
        {
            m=(m*r)%p;
            m=(m*s)%p;
        }
    
    return (String DQuery_result);
}
```

**Figure 4.8: Query Decryption Module in AQE–PSQLIA**

In the query decryption model, each character is read from the encrypted query and applies decryption. The coding module for query decryption is given in Figure 4.8.

User Validation Module

```java
Public void user_validation( )
{
    //Retrieve user name and password
    if (user_name= = username && password= = userpassword)
    {
        Then accept the query validity=TRUE;
    }
    else
    {
        reject the query validity=FALSE;
    }
    return(validity);
}
```

**Figure 4.9: User Validation Module in AQE–PSQLIA**
Validity of the username and password is matched with the user account database. If the user is a legitimate user then it returns TRUE else return FALSE. The sample coding for the user validation module is given in Figure 4.9.

4.2.3 Implementation Detail of the Proposed AQE–PSQLIA Scheme

This section provides an implementation detail for the proposed scheme with simple numerical example. Four phases (key generation, query generation, encryption and decryption) are considered for implementation. The key generation phase is used to generate the key pair for every user. Query generation phase is used to generate the SQL query using username and password. Query is encrypted by using encryption phase in client side. The query is decrypted by using decryption phase in server side.

Key Generation

Server selects the prime $p=2357$ and a generator $g=2$ of $Z_{2357}^*$. Server chooses the private key $x = 1751$ and computes

$$y = g^x \mod p = 2^{1751} \mod 2357 = 1185$$

Server’s key pair is $(p = 2357, g = 2, y = 1185)$

Query Generator Model

In this model query is generated by using user’s input. Simple example is given below.

```
SELECT * FROM User_Account WHERE User_name='Raja' AND Password='Bhuvana';
```

Encryption Model

Each character is read from the SQL query and encrypts as shown in the Figure 4.10.

```
SELECT * FROM User_Account WHERE User_name='Raja' AND Password='Bhuvana'
```

Figure 4.10: Token Separation in SQL Query
This model illustrates the encryption process for a single character “S”.
Step 1: Select a random \( k \) value \( (k=15) \)
Step 2: ASCII value of S is 83 (\( M=83 \))
Step 3: Encrypt S using public key \( y = 1185 \) of server
Step 4: Compute \( r = g^k \mod p \), Where \( g=2, \ r = 2^{15} \mod 2357 = 2127 \)
Step 5: Compute \( s = M^{y^k} \mod p \), Where \( M=83, \ s = 83.1185^{15} \mod 2357 = 902 \)
Step 6: Cipher text is \( (r, s) = (2127, 902) \)
   In encryption model, these steps are applied for each character from the SQL query string.

**Decryption Model**
In the decryption model, each character is decrypted as follows
Step 1: Compute \( M = r^{-y} \mod p = 2127^{-1751} \cdot 902 \mod 2357 = 541.902 \mod 2357 = 83 \)
Step 2: \( M=83 \) and equivalent character is “S”

**4.2.4 Security Analysis of the Proposed Scheme**
This section provides a security analysis of the proposed scheme. The proposed scheme is secured against well known SQLIA ’s. Security analysis of the proposed scheme is discussed with a SQLIA scenario as shown in Figure 4.11.

**Assumption**
*Scenario*: Assume that an attacker captures the login request sent by the user to server and decrypts the message by using user public key.

**Figure 4.11: Attacking Scenario in AQE--PSQLIA**

95
Argument

Attack: Let us assume that attacker captures the login query and tries to break the encryption algorithm by applying any key breaking schemes. This is not possible in the proposed scheme because ElGamal’s security rests on the difficulty of solving the discrete logarithm problem in $Z_p$ (see the proof).

Proof:

In general, let $Z_p$ be a finite cyclic group with $p$ elements. Let $g$ be a generator of $Z_p$; then the public key $Y_S$ is generated by using private key $X_S$ of server and this can be written in the form $Y_S = g^{X_S} \mod p$. No such algorithm can compute efficiently calculate $X_S$ from known $g$, $p$ and $Y_S$. Therefore, proposed scheme is secure against well known encryption breaking schemes.

4.3 AUTHENTICATION MECHANISM TO PREVENT SQL INJECTION ATTACK (AM–PSQLIA)

In this section, an authentication mechanism is proposed to prevent SQL Injection Attack. This method uses Advanced Encryption Standard (AES) for user authentication.

Contribution of the Proposed Scheme:

1. In the proposed scheme, Advance Encryption Standard (AES) is used for encrypting username and password. This will introduce more security in web browser level.
2. Static Query Checker is used to identify the suspicious character in the user input.
3. Computational cost for the proposed scheme is less and it is negligible. It provides high security by applying cryptographic algorithms.

4.3.1 Proposed AM–PSQLIA Scheme

This section proposed an authentication scheme using AES for preventing SQL Injection Attack. This method has three phases: Registration Phase, Query Generation Phase and Verification Phase (detection and prevention phase).
Registration Phase in AM–PSQLIA

The following steps are executed, whenever a new user enters into the server for registering as a new user.

1. Every user must select a unique Username $U_{name}$ and Password $U_{password}$ and send it to the server along with registration request.
2. Server receives the request from the user and registers as a new user. Server maintains a user account table with three field’s Username, User Password, and User Secret Key (unique key value) as shown in the Table 4.1. The user secrete key $K_{U_{name}}$ is generated by the server and this key is unique for all the users.
3. Server sends a registration conformation to the user along with user secrete key $K_{U_{name}}$.

<table>
<thead>
<tr>
<th>User Name</th>
<th>User Password</th>
<th>User Secret Key</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vidhya</td>
<td>Bala</td>
<td>$SecretKey_{Vidhya}$</td>
</tr>
<tr>
<td>Amutha</td>
<td>Prabakar</td>
<td>$SecretKey_{Amutha}$</td>
</tr>
<tr>
<td>Jancy</td>
<td>Prathima</td>
<td>$SecretKey_{Jancy}$</td>
</tr>
</tbody>
</table>

Query Generation Phase in AM–PSQLIA

In this phase, SQL query is generated by using encrypted username and password the following steps are involved in query generation phase:

1. The user data is validated by using wild card checker module. The static wild card characters are used for detecting malicious characters from user input.
2. The username and password is encrypted by applying Advanced Encryption Standard using users secrete key.
3. SQL query generator generates the query by using the encrypted username and password send to the server as shown in Figure 4.12.
Query_result=SELECT * FROM user_account WHERE username = 'Amutha'
AND Password = 'Prabakar' AND encrypted_username = 'E_{SecretKey} (Amutha)'
AND encrypted_password = 'E_{SecretKey} (Prabakar)'

Figure 4.12: Sample Query Generation in AM–PSQLIA

Verification Phase in AM–PSQLIA

In the verification phase (detection and prevention phase), server receives
the query result sent by the user and performs the following steps:
a. The server receives the query and decrypts the Username and Password
   by using user’s secret key.
b. Check the decrypted Username and Password from the user account
   table. If it matches then accept the user as a valid user, otherwise reject
   as a malicious attacker.

The detailed system model of query generation and verification phase
(detection and prevention phase) is given in Figure 4.13. In the proposed AM–
PSQLIA scheme, verification phase first decrypt the username from the query and corresponding secret key of the user is taken from user account table. If the given user input is like username= ‘a’ or ‘1’= ‘1’; -- then the SQL Injection Attack is prohibited directly in wild-card checker module itself. If the given user name is not valid then the user is prohibited as illegitimate.

4.3.2 Coding Model for Proposed AM–PSQLIA

This section illustrates the coding model for the proposed scheme. The coding part is categorized into five modules as shown in the above Figure 4.13.

- Wild-Card Checker Model
- Username and Password Encryption Module(Figure 4.14)
- SQL Query Generator Module(Figure 4.15)
- Decryption Module(Figure 4.16)
- User Validation Module(Figure 4.17)

Wild-Card Checker Module

In this module, the user input is validated by using wild-card checker, which is statically generated in application server. The coding model for the wild-card checker is given in section 4.2.3.

User name and Password Encryption Module

In this phase the username and password is encrypted using AES. User input is read as character by character and encrypt the character using user secret key. The coding model of the username and password encryption module is given below (Figure 4.14).

```java
public class AES {
    public static String asHex (byte buf[])
    {
        StringBuffer strbuf = new StringBuffer(buf.length * 2);
        int i;
        for (i = 0; i < buf.length; i++)
        {
            // encryption code
        }
    }
}
```
```java
{
    if (((int) buf[i] & 0xff) < 0x10)
        sb.append("0");
    sb.append(Long.toString((int) buf[i] & 0xff, 16));
}
return sb.toString();
}
public void encrypt(String S) {
    // Instantiate the cipher
    Cipher cipher = Cipher.getInstance("AES");
    cipher.init(Cipher.ENCRYPT_MODE, skeySpec);
    byte[] encrypted = cipher.doFinal((args.length == 0 ? "Bhuvana" :
                                        args[0]).getBytes());
    return (encrypted);
}
```

**Figure 4.14: Username and Password Encryption in AM–PSQLIA**

**SQL Query Generator Module**

In the query generator module, the login query is generated by using the encrypted username and password. The coding model is given in Figure 4.15.

```java
System.out.println("Enter User Name:");
name = in1.nextLine(); // Read one line from the console.
System.out.println("Enter Password:");
password = in1.nextLine();
Ename=AES.encrypt(name); //Encrypt userID using AES
Epassword=encrypt(password); //Encrypt user password
String Query_result="SELECT * FROM user_account WHERE 
user_name="+name+" AND password="+password+" AND E_username 
= "+Ename+" AND E_password = "+Epassword+";
```

**Figure 4.15: SQL Query Generator Module in AM–PSQLIA**
**User Name and Password Decryption Module**

The server receives the query and decrypts the username and password by using user secret key. The encrypted username and password is read as character by character and apply decryption. The coding model of the username and password decryption module is given below (Figure 4.16).

```java
public class AES {
    public static String asHex (byte buf[]) {
        StringBuffer sbuff = new StringBuffer(buf.length * 2);
        int i;
        for (i = 0; i < buf.length; i++) {
            if (((int) buf[i] & 0xff) < 0x10)
                sbuff.append("0");
            sbuff.append(Long.toString((int) buf[i] & 0xff, 16));
        }
        return sbuff.toString();
    }
    public void decrypt(String S)
    {
        cipher.init(Cipher.DECRYPT_MODE, skeySpec);
        byte[] original = cipher.doFinal(S);
        String originalString = new String(original);
        return (original)
    }
}
```

**Figure 4.16: Username and Password Decryption in AM–PSQLIA**

**User Validation Module**

The data are retrieved from the user account database and compared with the decrypted username and password. Coding model is given in Figure 4.17
Public void user_validation()  
{  
    Username=decrypt(Eusername);  
    Dpassword=decrypt(Epassword);  
    if (user_name= = Username && password= = Duserpassword)  
    {Then accept the query  
        validity=TRUE;  
    }  
    else  
    { reject the query  
        validity=FALSE;  
    }  
    return(validity);  
}  

Figure 4.17: User Validation Phase in AM–PSQLIA  

4.3.3 Implementation Detail for Proposed AM–PSQLIA  
This section provides an implementation detail for three phases (Username and Password encryption using AES, Query generation phase, Username and Password decryption phase) with simple numerical example.  

Username and Password Encryption using AES  
In this model, username and the password are encrypted by using user’s secret key. For example, username is “raja” and password is “bhuvana”. The username and the password are encrypted and the encrypted username as “3373a8ef5”and password as 319a8507”.  

Query Generation Phase  
In this model query is generated by using user’s input. Simple example is given below.

```
SELECT * FROM User_Account WHERE User_name=’Raja’ AND Password=’Bhuvana’ AND E_username = ‘3373a8ef5’ AND E_password = ‘319a8507’;
```
**User Name and Password Decryption Phase**

In this phase, server receives the query and decrypts the encrypted username (E_username) and password (E_password) by using user secrete key. If the decrypted username and password match with the confidential data in the user account table, which is maintained by the database server, then the user is considered as a legitimate user.

Advanced Encryption Standard (AES) is very secure symmetric key encryption scheme; even side channel attack is not possible to break AES. The proposed AM–PSQLIA is very secure, because AES is used for username and password encryption. The proposed technique almost prevents all the malicious queries.

### 4.4 AUTHENTICATION SCHEME FOR PREVENTING SQL INJECTION ATTACK USING HYBRID ENCRYPTION

This section proposed an authentication scheme using hybrid encryption algorithm - the combination of Advance Encryption Standard (AES) and RSA for preventing SQL Injection Attack. In the proposed scheme, unique secret key is assigned for every user in AES encryption and the server has private key and public key pair for RSA encryption. In the proposed method two levels of encryption is used for query encryption.

a. Apply symmetric key encryption for encrypting username and password by using User Secret Key

b. Apply asymmetric key encryption for query encryption by using server’s public key.

The proposed scheme is highly secured due to applying this hybrid encryption algorithm. This method has three phases: Registration Phase, Login Phase and Verification Phase (detection and prevention phase).
Contribution of the Proposed Scheme:

1. In the proposed scheme, Advance Encryption Standard (AES) is used for encrypting username and password, RSA is used for query encryption.
2. In this method, attacker cannot introduce suspicious characters in the query. If the attacker introduces the suspicious characters in the user data this will be identified in wild card checker module and discard it as malevolent.
3. The proposed scheme needs high computational cost due to application of hybrid encryption.

4.4.1 Proposed PSQLIA-HE Scheme

The proposed PSQLIA-HE scheme has three phases: Registration phase, Login phase and Verification phase (detection and prevention phase). The phases are explained below:

Registration Phase in PSQLIA-HE

The following steps are executed, when ever a new user enters into the server to register as a new user:

a. Every user must select a unique username $U_{Name}$ and password $U_{Password}$ and send it to the server along with registration request.

b. Server receives the request from the user and registers as a new user. Server maintains a user account table with three field’s username, user password and user secrete key (unique key value) as shown in the Table 4.2. The user secrete key $k_{U_{Name}}$ is generated by the server and this key is unique for all the users.

c. Server sends a registration conformation to the user along with user secrete key $k_{U_{Name}}$.

The system model for the registration phase is given below as shown in Figure 4.18.
Table 4.2: User Account Table in PSQLIA-HE

<table>
<thead>
<tr>
<th>User Name</th>
<th>User Password</th>
<th>User Secret Key</th>
</tr>
</thead>
<tbody>
<tr>
<td>Murugesh</td>
<td>Uthandakalai</td>
<td>(\text{SecretKey}_{\text{Murugesh}})</td>
</tr>
<tr>
<td>Raja</td>
<td>Sekar</td>
<td>(\text{SecretKey}_{\text{Raja}})</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

Figure 4.18: System Model for User Registration Phase in PSQLIA-HE

Login Phase in PSQLIA-HE

Through login phase user can access the database from the server and the following steps are performed in login phase.

a. The username and password are encrypted by using Advanced Encryption Standard algorithm using user secrete key.

b. SQL query generator generates the query by using the encrypted username and password as shown in Figure 4.19.

c. The query result is encrypted by using RSA cryptosystem by applying server’s public key as shown in Figure 4.20. The encrypted query will be sent to server.

```sql
Query_result=SELECT * FROM user_account WHERE username = ‘Raja’ AND password = ‘Sekar’ AND E_username = ‘E_{\text{SecretKey}_{\text{Raja}}}’ AND E_password = ‘E_{\text{SecretKey}_{\text{Sekar}}}’
```

Figure 4.19: Sample Query Generation in PSQLIA-HE
Figure 4.20: Encrypted Query Result in PSQLIA-HE

Verification Phase in PSQLIA-HE

a. The server receives the encrypted query and decrypts the query by using server’s private key. Decrypt the username and password from the query by using secret key of user.

b. Check the decrypted username and password from the user account table. If it matches then the user is considered as a legitimate user, otherwise malicious user. The detailed system model of login and verification phase is given in Figure 4.21.

Figure 4.21: Login and Verification Phase in PSQLIA-HE

4.4.2 Coding Model for Proposed PSQLIA-HE

This section illustrates the coding model for the proposed PSQLIA-HE scheme.
The coding part is categorized into seven modules given below:

- Wild-Card Checker
- Username and Password Encryption
- SQL Query Generator
- SQL Query Encryption
- SQL Query Decryption
- Username and Password Decryption
- User Validation

**Wild-Card Checker Module**

In this module, the user input is validated by using wild-card checker, which is statically generated in application level. The coding model for the wild-card checker is given in section 4.2.3.

**Username and Password Encryption**

In this module the username and password is encrypted using AES. User input is read as character by character and encrypts the character by using user secret key. The coding model of the username and password encryption phase is given below (Figure 4.22).

```java
public class AES {
    /*Turns array of bytes into string param buf   Array of bytes to convert to hex string
    return Generated hex string     */
    public void encrypt(String S)
    {
        // Instantiate the cipher
        Cipher cipher = Cipher.getInstance("AES");
        cipher.init(Cipher.ENCRYPT_MODE, skeySpec);
        byte[] encrypted = cipher.doFinal((args.length == 0 ? "Bhuvana" : args[0]).getBytes());
        return (encrypted);
    }
}
```

*Figure 4.22: Username and Password Encryption Module in PSQILIA-HE*
SQL Query Generation

This phase generates the SQL query based on encrypted username and password. The sample coding for SQL query generation module is shown in Figure 4.23.

```java
//... Read Input from user
System.out.println("Enter User Name:");
name = in1.nextLine(); // Read one line from the console.
System.out.println("Enter Password:");
Password = in1.nextLine();
in.close(); //Note 2
Ename=AES.encrypt(name); //Encrypt userID using AES by applying user secrete key
EPassword=encrypt(Password); //Encrypt user Password

//... Display Query Result
String Query_result="SELECT * FROM user_account WHERE
user_name="+name+" AND password="+Password+" AND
E_username = “+Ename+” AND
E_password = “+EPassword+”;
```

Figure 4.23: SQL Query Generation Module in PSQLIA-HE

SQL Query Encryption

In the Query encryption module, each character is read from the SQL query and applies encryption by using RSA cryptosystem. The sample code for SQL query encryption module is given in Figure 4.24.
String Encryptor( String *Query_result, int p, int y, int g)
{
    //… Characters are separated from query and apply encryption
    for (int j=0;j<Query_result.length();j++){
        Encrypted_Query[j]=RSA.Encrypt(Query_result[j])
    } Return (String Encrypted_Query);
}

Figure 4.24: SQL Query Encryption Module in PSQLIA-HE

SQL Query Decryption

In the query decryption module, each character is read from the encrypted query and applies decryption. The sample code for SQL query decryption module is given in Figure 4.25.

String Decryptor( String *Encrypted_Query, int p, int y, int g)
{
    //… Characters are separated from query and apply decryption
    for (int j=0;j<Encrypted_Query.length();j++)
    {
        Original_Query[j]=RSA.Decrypt(Encrypted_Query[j])
    } Return (String Original_Query);
}

Figure 4.25: SQL Query Decryption Module in PSQLIA-HE

Username and Password Decryption

The server decrypts the username and password by using secret key by server. The encrypted username and password is read as character by character and apply decryption. The coding model of the username and password decryption module is given in Figure 4.26.
public class AES {
    /* Turns array of bytes into string  param buf  Array of bytes to convert to hex string
    return  Generated hex string */
    public static String asHex (byte buf[]) { 
        for (i = 0; i < buf.length; i++) {
            if ((((int) buf[i] & 0xff) < 0x10) strbuf.append("0");
                strbuf.append(Long.toString((int) buf[i] & 0xff, 16));
        } return strbuf.toString(); }
    public void decrypt(String S) {
        cipher.init(Cipher.DECRYPT_MODE, skeySpec);
            System.out.println("Original string: "+originalString + "+ asHex(original));
                return (original) }
    }

    Figure 4.26: Username and Password Decryption Module in PSQLIA-HE
    User Validation Model

    The data’s are retrieved from the user account database and compared with
    the decrypted username and password. The sample code for user validation model
    is given in Figure 4.27.

    public void user_validation() {
        DUsername=decrypt(EUsername);
        DPassword=decrypt(EPassword);
        if (user_name= = DUsername && password= = DUserpassword)
            {Then process the query
            validity=TRUE;
            } else { reject the query
        validity=FALSE;
            } return(validity);
    }

    Figure 4.27: User Validation Module in PSQLIA-HE
4.4.3 Implementation Detail for Proposed PSQLIA-HE

This section provides an implementation detail for five (Key Generation for RSA, Username and Password Encryption using AES, Query Generator Module, Encryption Module for RSA and Decryption Module for RSA) phases of the proposed scheme with simple numerical example.

Key Generation for RSA

Server selects the prime \( p=2357 \) and \( q=2551 \). Computes \( n=pq=6012707 \) and \( \phi(n)=(p-1)(q-1)=6007800 \). Server chooses \( e=3674911 \), using Extended Euclidean Algorithm, compute \( d=422191 \), such that \( ed \equiv 1 \pmod{\phi(n)} \). Server’s public key is the pair of \( (n=6012707,e=3674911) \), while server’s private key is \( d=422191 \).

User name and Password Encryption using AES

In this module, username and password are encrypted by using user’s secret key. For example, username is “raja” and Password is “bhuvana”. The username and password is encrypted and the encrypted username and password is “3373a8ef5” and “319a8507”.

Query Generator Module

This module generates the query by using encrypted username and password. The example for query generation model is given below:

```
SELECT * FROM User_Account WHERE User_name='Raja' AND Password='Bhuvana' AND E_username = 'E_SecretKey (Raja)' AND E_password = 'E_SecretKey (Bhuvana)';
```

Encryption Module for RSA

```
SELECT * FROM User_Account WHERE User_name='Raja' AND Password='Bhuvana' AND E_username = 'Encrypt(Raja)' AND E_password = 'Encrypt(Bhuvana)';
```

Figure 4.28: SQL Query Tokens in PSQLIA-HE
Each character is read from the SQL query as shown in Figure 4.28 and encrypts the tokens. This model illustrates the encryption process for a single character “S”.

Step 1: ASCII value of S is 83 (M=83)
Step 2: Encrypt S using public key \( (n = 6012707, e = 3674911) \) of server
Step 3: Compute \( C = M^e \mod n \), Where \( M=83 \), \( C = 83^{3674911} \mod 6012707 = 3453768 \)
Step 4: Cipher text is “3453768”. In encryption model, same set of steps are applied for each character from the SQL query string.

**Decryption Model for RSA**

In the decryption model, each character is decrypted as follows

Step 1: Compute \( M = C^d \mod n = 3453768^{422191} \mod 6012707 = 83 \)
Step 2: M=83 and equivalent character is “S”.

**4.4.4 Security Analysis of the Proposed PSQLIA-HE Scheme**

The proposed scheme is secure against well known SQL Injection Attack. Security analysis of the proposed scheme is discussed with a SQL Injection Attack scenario given in Figure 4.11.

**Assumption**

*Scenario:* Assume that an attacker captures the query send by the user to server and decrypts the message by using user public key.

**Argument**

*Attack:* Let us assume that attacker captures the query and tries to break the encryption algorithm by applying some key breaking schemes. This is not possible in the proposed scheme because RSA security rests on the difficulty of solving the integer factorization problem (see the proof)

*Proof:* The security of the RSA cryptosystem is based on two mathematical problems: the problem of factoring large numbers and the RSA problem. Full decryption of an RSA ciphertext is thought to be infeasible on the assumption that both of these problems are hard, i.e., no efficient algorithm exists for solving these problems.