



الجمهورية العربية السورية

جامعة دمشق

كلية الهندسة المعلوماتية

قسم هندسة البرمجيات ونظم المعلومات

[Date]

[Task2]

[ATM simulation application]

عمل الطلاب

وئام زهير بريك هنيدي رغد وائل نصر

اشراف المهندسة:

اريج رحال

مقدمة:

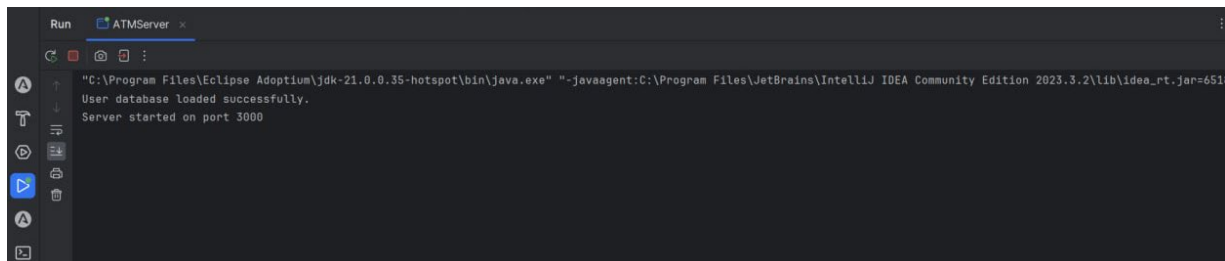
تطبيق `ATMServer` و `ATMClient` هو تطبيق خادم و عميل محاكاة لنظام ماكينة الصراف الآلي (ATM) باستخدام تقنيات التشفير لتحسين الأمان. يقوم التطبيق بخدمة مجموعة من الوظائف المرتبطة بإدارة الحسابات البنكية للمستخدمين مثل التسجيل، تسجيل الدخول، الإيداع، السحب، والتحقق من الرصيد.

تشغيل ال Server:

يبدأ الخادم في الاستماع على المنفذ 3000 باستخدام `ServerSocket`. هذه الخطوة تعني أن السيرفر سيبدأ في الاستماع لأي اتصالات واردة من عملاء على هذا المنفذ.

```
ServerSocket serverSocket = new ServerSocket (PORT);
System.out.println("Server started on port " + PORT);

// Continuously accept and handle client connections
while (true) {
    Socket clientSocket = serverSocket.accept();
    System.out.println("Client connected: " + clientSocket.getInetAddress(). getHostAddress ());
    new ClientHandler (clientSocket, userDatabase). start ();
}
```



:Signup

يستطيع المستخدم إضافة ال Username وال password ويتم التواصل مع ال server ويتم تخزينهم ب text file بالإضافة الى إضافة الرصيد بقيمة صفر

- **ATMClient:**

case 2: // Register

```

System.out.print("Enter a new username: ");
String newUsername = scanner.nextLine(); // Get the new username
System.out.print("Enter a new password: ");
String newPassword = scanner.nextLine(); // Get the new password

// Encrypt the username and password using RSA public key
String encryptedNewUsername = RSAUtil.encrypt(newUsername, serverPublicKey);
String encryptedNewPassword = RSAUtil.encrypt(newPassword, serverPublicKey);
System.out.println(encryptedNewUsername);
// Send registration request and credentials to the server
out.writeObject("REGISTER");
out.writeObject(encryptedNewUsername);
out.writeObject(encryptedNewPassword);

// Read the server's response for registration success or failure
boolean registrationSuccessful = (Boolean) in.readObject();
if (registrationSuccessful) {
    System.out.println("Registration successful! You can now log in.");
} else {
    System.out.println("Registration failed. Username may already exist.");
}
break;

```

يتم تشفير البيانات المرسلّة باستخدام خوارزمية ال RSA
حيث أول ماتم تشغيل السيرفر يقوم بارسال public key to client

- **ATMServer:**

```

public void run() {
    try (ObjectInputStream in = new ObjectInputStream(socket.getInputStream());
        ObjectOutputStream out = new ObjectOutputStream(socket.getOutputStream())) {

        // Send the server's RSA public key to the client
        out.writeObject(keyPair.getPublic());
        System.out.println("Sent public key to client.");
    }
}

```

يقوم ال Client باستقبال ال public key

- **ATMClient:**

```
serverPublicKey = (PublicKey) in.readObject();
```

ثم يقوم بالتشفير باستخدام خوارزمية ال RSA حيث يتم إعطاء الخوارزمية ال publickey وال plaintext ويتم ارسال القيم مشفرة

- **ATMClient:**

```
// Encrypt the username and password using RSA public key
String encryptedUsername = RSAUtil.encrypt(username, serverPublicKey);
String encryptedPassword = RSAUtil.encrypt(password, serverPublicKey);

// Send login request and credentials to the server
out.writeObject("LOGIN");
out.writeObject(encryptedUsername);
out.writeObject(encryptedPassword);
```

:RSA

يستخدم RSA لتأمين تبادل المفاتيح (Key Exchange) بين العميل والخادم.

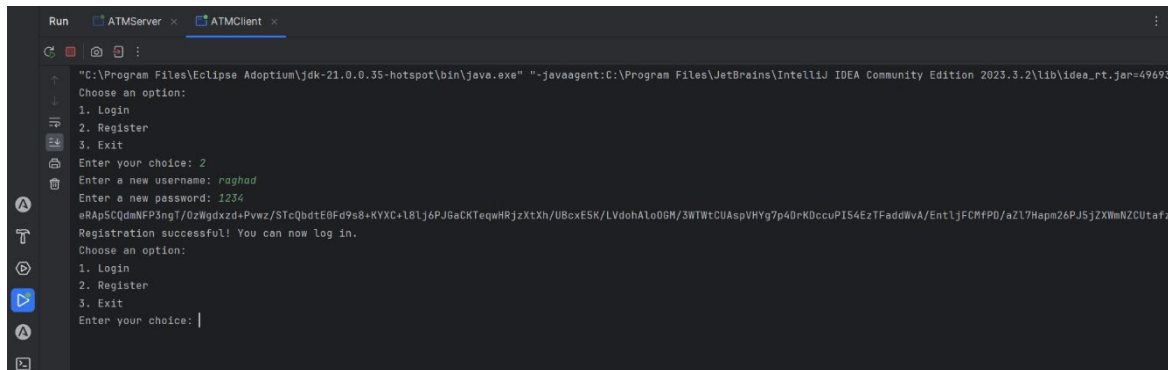
RSA يعتمد على مفتاحين: مفتاح عام للتشفير ومفتاح خاص لفك التشفير.

في التطبيق، يتم استخدام المفتاح العام من قبل العميل لتشفير البيانات (مثل اسم المستخدم وكلمة المرور) بحيث لا يمكن فك تشفيرها إلا باستخدام المفتاح الخاص الموجود فقط على الخادم.

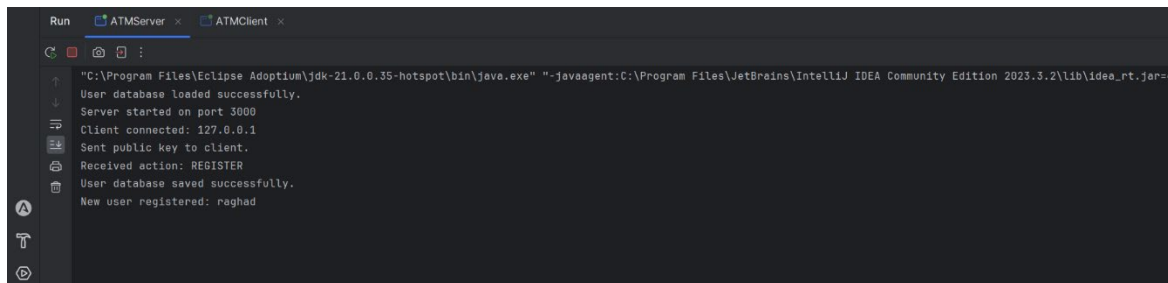
هذا يمنع أي طرف ثالث من اعتراض البيانات وفك تشفيرها.

```
import javax.crypto.*;
import java.security.*;
import java.util.Base64;
public class RSAUtil {
    // Encrypt using the RSA public key
    public static String encrypt(String data, PublicKey publicKey) throws Exception {
        Cipher cipher = Cipher.getInstance("RSA");
        cipher.init(Cipher.ENCRYPT_MODE, publicKey);
        byte[] encryptedBytes = cipher.doFinal(data.getBytes());
        return Base64.getEncoder().encodeToString(encryptedBytes);
    }
    // Decrypt using the RSA private key
    public static String decrypt(String data, PrivateKey privateKey) throws Exception {
        byte[] encryptedBytes = Base64.getDecoder().decode(data);
        Cipher cipher = Cipher.getInstance("RSA");
        cipher.init(Cipher.DECRYPT_MODE, privateKey);
        byte[] decryptedBytes = cipher.doFinal(encryptedBytes);
        return new String(decryptedBytes);
    }
    // Generate a key pair (public and private)
    public static KeyPair generateKeyPair() throws NoSuchAlgorithmException {
        KeyPairGenerator keyPairGenerator = KeyPairGenerator.getInstance("RSA");
        keyPairGenerator.initialize(2048);
```

```
return keyPairGenerator.generateKeyPair();}}
```



```
Run  ATMServer x  ATMClient x
"C:\Program Files\Eclipse Adoptium\jdk-21.0.0.35-hotspot\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2023.3.2\lib\idea_rt.jar=49693
Choose an option:
1. Login
2. Register
3. Exit
Enter your choice: 2
Enter a new username: raghad
Enter a new password: 1234
eRApSCQdmNFP3ngT/0zWgdxzd+Pwz/STcQbdtE0Fd9s8+KYXC+L8lj6PJGaCKTeqWHRjzXtXh/UBcxESK/LVdohAlo0GH/3WTWtCUAspVHYg7p40rKDccuPI54EzTFaddWvA/EntljFCMfPD/a2l7Hapm26PJ5jZXWmNZCÚtafz
Registration successful! You can now log in.
Choose an option:
1. Login
2. Register
3. Exit
Enter your choice: |
```



```
Run  ATMServer x  ATMClient x
"C:\Program Files\Eclipse Adoptium\jdk-21.0.0.35-hotspot\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2023.3.2\lib\idea_rt.jar=4
User database loaded successfully.
Server started on port 3000
Client connected: 127.0.0.1
Sent public key to client.
Received action: REGISTER
User database saved successfully.
New user registered: raghad
```

يقوم السيرفر باستقبال القيم وفك تشفيرها باستخدام خوارزمية ال RSA ويتم تخزين القيم بال file
لتخزين القيم بال file يتم استدعاء خوارزمية التشفير المتناظر AES
حيث يتم تخزين الاسم بدون تشفير اما كلمة المرور والرصيد يتم تشفيره ثم تخزينه مشفر

o **ATMServer:**

```
private void handleRegister(ObjectInputStream in, ObjectOutputStream out) throws Exception {
    String encryptedUsername = (String) in.readObject();
    String encryptedPassword = (String) in.readObject();
    // Decrypt the username and password using the server's RSA private key
    String username = RSAUtil.decrypt(encryptedUsername, keyPair.getPrivate());
    String password = RSAUtil.decrypt(encryptedPassword, keyPair.getPrivate());
    // Check if the username already exists
    if (userDatabase.containsKey(username)) {
        System.out.println("Registration failed: Username " + username + " already exists.");
        out.writeObject(false); // Registration failed
    } else {
        // Encrypt the password and initialize the balance
        String encryptedPasswordAES = AESUtil.encrypt(password, secretKey);
        String encryptedBalance = AESUtil.encrypt("0.0", secretKey);
```

```
// Store the new user data in the database
userDatabase.put(username, encryptedPasswordAES + "," + encryptedBalance);
saveUserDatabase(userDatabase);
System.out.println("New user registered: " + username);
out.writeObject(true); // Registration successful
```

:AES

AES سريع وفعال مقارنة بـ RSA عند التعامل مع كميات كبيرة من البيانات.

يتم استخدامه لتشفير البيانات الحساسة (مثل كلمات المرور وأرصدة الحسابات) المخزنة في قاعدة البيانات.

بما أن AES يعتمد على مفتاح واحد مشترك، فإنه أسرع وأقل استهلاكًا للموارد مقارنة بـ RSA، وهو مناسب لتشفير البيانات أثناء التخزين أو أثناء العمليات الداخلية.

الجمع بينهما:

RSA يُستخدم لتأمين تبادل المفتاح السري لـ AES.

AES يُستخدم بعد ذلك لتشفير وفك تشفير البيانات بشكل سريع أثناء تخزينها أو معالجتها.

هذا التصميم يوفر أمانًا عاليًا وأداءً جيدًا، حيث أن RSA يوفر أمانًا قويًا أثناء نقل المفاتيح، وAES يضمن التشفير السريع للبيانات

مفتاح التشفير نفسه مفتاح فك التشفير حيث يتم استخدام مفتاح سري (Secret key)

التشفير (Encryption): يتم تحويل البيانات الأصلية إلى نص مشفر باستخدام AES مع مفتاح سري، ثم يتم تحويل البيانات المشفرة إلى تنسيق قابل للإرسال باستخدام Base64.

فك التشفير (Decryption): يتم عكس العملية باستخدام نفس المفتاح السري لفك تشفير البيانات المستلمة (التي تم تمثيلها في Base64).

```
import javax.crypto.Cipher;
import javax.crypto.SecretKey;
import javax.crypto.spec.SecretKeySpec;
import java.util.Base64;

public class AESUtil {

    private static final String ALGORITHM = "AES";

    public static String encrypt(String data, SecretKey key) {
        try {
            Cipher cipher = Cipher.getInstance(ALGORITHM);
            cipher.init(Cipher.ENCRYPT_MODE, key);
            byte[] encryptedData = cipher.doFinal(data.getBytes());
            return Base64.getEncoder().encodeToString(encryptedData);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

```

    } catch (Exception e) {
        throw new RuntimeException("Error while encrypting data: " + e.getMessage());
    }
}

public static String decrypt(String encryptedData, SecretKey key) {
    try {
        Cipher cipher = Cipher.getInstance(ALGORITHM);
        cipher.init(Cipher.DECRYPT_MODE, key);
        byte[] decodedData = Base64.getDecoder().decode(encryptedData);
        byte[] originalData = cipher.doFinal(decodedData);
        return new String(originalData);
    } catch (Exception e) {
        throw new RuntimeException("Error while decrypting data: " + e.getMessage());
    }
}

```

File Text:

```

Project: Security
  - idea
  - out
  - AESUtil
  - ATMClient
  - ATMServer
  - RSAUtil
  - user_data.txt
  - External Libraries
  - Scratches and Consoles

Run: ATMClient
  1. Login
  2. Register
  3. Exit
  Enter your choice: 2
  Enter a new username: raghad
  Enter a new password: 1234
  mQLoURVNuMpi+7UiyosF8RiCCiqfU8WVqL0T1QM2N2MoZBh0ZFVMSasvSMTGxamkVH98Wv80Q2dLz0cMdqUu/90YxyQefpPYAVf5V2SqHS1phZ5U9A1Cbesl6oueHwDqr3M977MyL04Cp1Mgenzkdlo+Cq6dWr8xJ5AJJfa6W5x81vz
  Registration successful! You can now log in.
  Choose an option:
  1. Login
  2. Register
  3. Exit
  Enter your choice: |
  
```

:Login

- **ATMClient:**

```
switch (choice) {
    case 1: // Login
        System.out.print("Enter username: ");
        String username = scanner.nextLine(); // Get the username
        System.out.print("Enter password: ");
        String password = scanner.nextLine(); // Get the password

        // Encrypt the username and password using RSA public key
        String encryptedUsername = RSAUtil.encrypt(username, serverPublicKey);
        String encryptedPassword = RSAUtil.encrypt(password, serverPublicKey);

        // Send login request and credentials to the server
        out.writeObject("LOGIN");
        out.writeObject(encryptedUsername);
        out.writeObject(encryptedPassword);

        // Read the server's response for login success or failure
        boolean loginSuccessful = (Boolean) in.readObject();
        if (loginSuccessful) {
            System.out.println("Login successful!");
            userMenu(scanner, out, in, username); // Enter the user menu after successful
        } else {
            System.out.println("Login failed. Please check your credentials.");
        }
        break;
}
```

login

يتم ارسال القيم مشفرة باستخدام ال RSA

- **ATMServer:**

```
private void handleLogin(ObjectInputStream in, ObjectOutputStream out) throws Exception {
    String encryptedUsername = (String) in.readObject();
    String encryptedPassword = (String) in.readObject();

    // Decrypt the username and password
    String username = RSAUtil.decrypt(encryptedUsername, keyPair.getPrivate());
    String password = RSAUtil.decrypt(encryptedPassword, keyPair.getPrivate());

    // Retrieve stored user data
    String storedData = userDatabase.get(username);
    if (storedData != null) {
```



```

String[] parts = storedData.split(",");
String decryptedPassword = AESUtil.decrypt(parts[0], secretKey);

// Verify password
if (decryptedPassword.equals(password)) {
    System.out.println("Login successful for user: " + username);
    out.writeObject(true); // Login successful
    return;
}
}
System.out.println("Login failed for user: " + username);
out.writeObject(false); // Login failed
}

```

يتم فك القيم المشفرة باستخدام RSA ثم يتم التحقق من الاسم وكلمة المرور

```

Run  ATMServer x  ATMClient x
Enter your choice: 2
Enter a new username: raghad
Enter a new password: 1234
mQLoIURVNuNpi+7U1yosF8R1CC1qFU8WVqL0T1QM2N2MoZBh0ZFVMaasv5MTGxamkVH98Wv80Q2dLz0cMdqUu/90YxyQafpPYAVf5V2SqHS1phZ5U9A1CbseL6oueHw0qr3M977MyL04Cp1MgenzkdLo+Cq6dWw8xJ5AJffa6W5x81
Registration successful! You can now log in.
Choose an option:
1. Login
2. Register
3. Exit
Enter your choice: 1
Enter username: raghad
Enter password: 1234
Login successful!

Choose an option:
1. Deposit money
2. Withdraw money
3. Check balance
4. Exit
Enter your choice:

```

```

Run  ATMServer x  ATMClient x
"C:\Program Files\Eclipse Adoptium\jdk-21.0.0.35-hotspot\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2023.3.2\lib\idea_rt.jar=49686:C:\
User database loaded successfully.
Server started on port 3000
Client connected: 127.0.0.1
Sent public key to client.
Received action: REGISTER
User database saved successfully.
New user registered: raghad
Received action: LOGIN
Login successful for user: raghad

```

:Deposit

يتم إرسال القيمة التي أريد إضافتها على الرصيد مشفرة باستخدام ال RSA

○ ATMClient:

```
case 1: // Deposit money
    System.out.print("Enter deposit amount: ");
    double depositAmount = scanner.nextDouble(); // Get the deposit amount
    scanner.nextLine(); // Consume the newline

    // Encrypt the username for secure communication
    String encryptedUsernameForDeposit = RSAUtil.encrypt(username, serverPublicKey);

    // Send the deposit request to the server
    out.writeObject("DEPOSIT");
    out.writeObject(encryptedUsernameForDeposit);
    out.writeObject(depositAmount);

    // Read the server's response with the updated balance
    Double newBalance = (Double) in.readObject();
    if (newBalance != -1) {
        System.out.println("Deposit successful! New balance: " + newBalance);
    } else {
        System.out.println("Error processing deposit.");
    }
    break;
```

○ ATMServer:

```
private void handleDeposit(ObjectInputStream in, ObjectOutputStream out) throws Exception {
    String encryptedUsername = (String) in.readObject();
    double depositAmount = (Double) in.readObject();

    // Decrypt the username
    String username = RSAUtil.decrypt(encryptedUsername, keyPair.getPrivate());

    // Retrieve and update balance
    double currentBalance = getUserBalance(username);
    if (currentBalance != -1) {
        double newBalance = currentBalance + depositAmount;
        updateUserBalance(username, newBalance);
    }
}
```

```

        System.out.println("Deposit successful for user: " + username + ". New balance: " +
newBalance);
        out.writeObject(newBalance); // Send the updated balance to the client
    } else {
        System.out.println("Deposit failed for user: " + username);
        out.writeObject(-1.0); // Error case
    }
}
}

```

1. قراءة البيانات:

- تقوم بقراءة اسم المستخدم المشفر ومقدار الإيداع من العميل.

2. فك تشفير اسم المستخدم:

- يتم فك تشفير اسم المستخدم باستخدام المفتاح الخاص RSA الخاص بالخادم.

3. استرجاع الرصيد الحالي:

- يتم استرجاع الرصيد الحالي للمستخدم باستخدام دالة `getUserBalance` التي تقرأ بيانات المستخدم المشفرة من قاعدة البيانات.

4. تحديث الرصيد:

- إذا تم العثور على المستخدم، يتم إضافة مقدار الإيداع إلى الرصيد الحالي وحساب الرصيد الجديد.

- ثم يتم تحديث الرصيد الجديد في قاعدة البيانات باستخدام دالة `updateUserBalance`.

5. إرسال النتيجة:

- إذا تم الإيداع بنجاح، يتم إرسال الرصيد الجديد إلى العميل.
- إذا فشل الإيداع (مثل عدم العثور على المستخدم)، يتم إرسال قيمة -1.0 كإشارة لوجود خطأ.

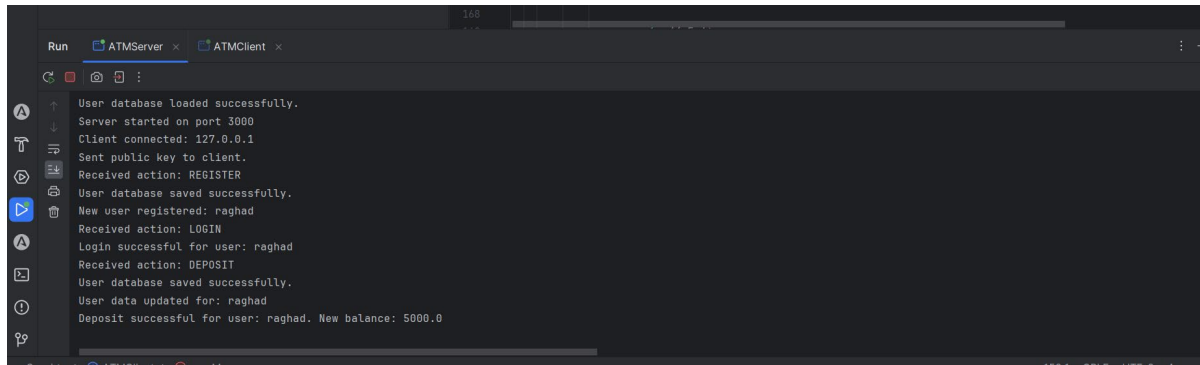
```

Run  ATMServer  ATMClient
Login successful!
Choose an option:
1. Deposit money
2. Withdraw money
3. Check balance
4. Exit
Enter your choice: 1
Enter deposit amount: 5000
Deposit successful! New balance: 5000.0

Choose an option:
1. Deposit money
2. Withdraw money
3. Check balance
4. Exit
Enter your choice: 1

```

Security > ATMServer > updateUserBalance 212:27 CRLF UTF-8 4 spaces



:UpdateUserBalance

```
private static void updateUserBalance(String username, double balance) {
    String storedData = userDatabase.get(username);
    String[] parts = storedData.split(",");
    String encryptedPassword = parts[0];
    String encryptedBalance = AESUtil.encrypt(String.valueOf(balance), secretKey);

    // Update the database
    userDatabase.put(username, encryptedPassword + "," + encryptedBalance);
    saveUserDatabase(userDatabase);
    System.out.println("User data updated for: " + username);
}
```

1. استرجاع البيانات المخزنة:

- يتم استرجاع بيانات المستخدم من قاعدة البيانات باستخدام اسم المستخدم (*username*).
- *storedData* تحتوي على كلمة المرور المشفرة والرصيد المشفر للمستخدم، مفصولين بفاصلة.

2. فصل البيانات:

- يتم تقسيم البيانات المخزنة إلى جزئين باستخدام *split(",")*: الجزء الأول يحتوي على كلمة المرور المشفرة (*encryptedPassword*)، والجزء الثاني يحتوي على الرصيد المشفر (*encryptedBalance*).

3. تشفير الرصيد الجديد:

- يتم تشفير الرصيد الجديد باستخدام طريقة *AESUtil.encrypt* لتحويل الرصيد إلى صيغة مشفرة.

4. تحديث قاعدة البيانات:

- يتم تحديث قاعدة البيانات بدمج كلمة المرور المشفرة والرصيد المشفر الجديد في المدخل الخاص بالمستخدم في *userDatabase*.

5. حفظ قاعدة البيانات:

- بعد تحديث البيانات في الذاكرة، يتم استدعاء الدالة *saveUserDatabase* لحفظ قاعدة البيانات المعدلة إلى الملف.

6. طباعة رسالة تأكيد:

○ يتم طباعة رسالة في وحدة التحكم لتأكيد أن بيانات المستخدم تم تحديثها بنجاح.

الهدف:

تحديث الرصيد المشفر للمستخدم في الذاكرة (قاعدة البيانات) وحفظ التعديلات في الملف لتجنب فقدان البيانات.

:Withdraw

○ **ATMClient:**

case 2: // Withdraw money

```
System.out.print("Enter withdrawal amount: ");
```

```
double withdrawAmount = scanner.nextDouble(); // Get the withdrawal amount
```

```
scanner.nextLine(); // Consume the newline
```

```
// Encrypt the username for secure communication
```

```
String encryptedUsernameForWithdraw = RSAUtil.encrypt(username, serverPublicKey);
```

```
// Send the withdrawal request to the server
```

```
out.writeObject("WITHDRAW");
```

```
out.writeObject(encryptedUsernameForWithdraw);
```

```
out.writeObject(withdrawAmount);
```

```
// Read the server's response with the updated balance
```

```
Double updatedBalance = (Double) in.readObject();
```

```
if (updatedBalance != -1) {
```

```
    System.out.println("Withdrawal successful! Remaining balance: " + updatedBalance);
```

```
} else {
```

```
    System.out.println("Insufficient balance or error.");
```

```

}
break;

```

○ **ATMServer:**

```

private void handleWithdraw(ObjectInputStream in, ObjectOutputStream out) throws
Exception {
    String encryptedUsername = (String) in.readObject();
    double withdrawAmount = (Double) in.readObject();

    // Decrypt the username
    String username = RSAUtil.decrypt(encryptedUsername, keyPair.getPrivate());

    // Retrieve and update balance
    double currentBalance = getUserBalance(username);
    if (currentBalance >= withdrawAmount) {
        double newBalance = currentBalance - withdrawAmount;
        updateUserBalance(username, newBalance);
        System.out.println("Withdrawal successful for user: " + username + ". New balance: " +
newBalance);
        out.writeObject(newBalance); // Send the updated balance to the client
    } else {
        System.out.println("Withdrawal failed for user: " + username + ". Insufficient funds.");
        out.writeObject(-1.0); // Insufficient funds
    }
}
}

```

1. استقبال البيانات:
 - تستقبل اسم المستخدم المشفر (encryptedUsername) والمبلغ المراد سحبه (withdrawAmount) من العميل.
2. فك تشفير اسم المستخدم:
 - يتم فك تشفير اسم المستخدم باستخدام المفتاح الخاص RSA.
3. استرجاع الرصيد الحالي:
 - يتم الحصول على الرصيد الحالي للمستخدم باستخدام دالة getUserBalance.
4. التحقق من كفاية الرصيد:
 - إذا كان الرصيد الحالي كافٍ للسحب (أكبر من أو يساوي المبلغ المطلوب):
 - يتم خصم المبلغ من الرصيد وتحديثه باستخدام updateUserBalance.

- إرسال الرصيد الجديد إلى العميل.
 - إذا لم يكن الرصيد كافٍ:
 - إرسال قيمة 1.0- إلى العميل كإشارة لفشل العملية بسبب نقص الأموال.
- الهدف من الدالة هو التعامل مع عمليات السحب والتحقق من كفاية الرصيد.

```

Run  ATMServer x  ATMClient x
1. Deposit money
2. Withdraw money
3. Check balance
4. Exit
Enter your choice: 2
Enter withdrawal amount: 3000
Withdrawal successful! Remaining balance: 2000.0

Choose an option:
1. Deposit money
2. Withdraw money
3. Check balance
4. Exit
Enter your choice: 1

```

```

Run  ATMServer x  ATMClient x
Received action: REGISTER
User database saved successfully.
New user registered: raghad
Received action: LOGIN
Login successful for user: raghad
Received action: DEPOSIT
User database saved successfully.
User data updated for: raghad
Deposit successful for user: raghad. New balance: 5000.0
Received action: WITHDRAW
User database saved successfully.
User data updated for: raghad
Withdrawal successful for user: raghad. New balance: 2000.0

```

:Check balance

○ ATMClient

case 3: // Check balance

// Encrypt the username for secure communication

String encryptedUsernameForBalance = RSAUtil.encrypt(username, serverPublicKey);

// Send the balance inquiry request to the server

out.writeObject("BALANCE");

out.writeObject(encryptedUsernameForBalance);

// Read the server's response with the current balance

Double balance = (Double) in.readObject();

System.out.println("Your current balance is: " + balance);

o ATMServer:

```
private static double getUserBalance(String username) {  
    String storedData = userDatabase.get(username);  
    if (storedData == null) return -1; // User not found  
    String[] parts = storedData.split(",");  
    String encryptedBalance = parts[1];  
    return Double.parseDouble(AESUtil.decrypt(encryptedBalance, secretKey));  
}
```

