



SECURE FILE TRANSFER APPLICATION

Design Manual

Author: Aoife O'Brien

Student ID: C00214279

Project Supervisor: Patrick Tobin

Recipient: Institute of Technology Carlow

Date: Friday 29th November 2019

Contents

Introduction..... 3

Front-end Layout 3

 Screen-flow Screenshots 3

System Architecture..... 7

Database..... 9

 Table Structure..... 9

 Users Table 9

Sockets 12

Encryption and Decryption 12

References..... 14

Introduction

This document will provide the reader with a thorough understanding of the work required to create this application. It will equip the reader with the knowledge to design the application described below, with a deep understanding of the internal architecture and user interface layout. This document will be broken into five sections; front-end, database, system architecture, sockets and encryption and decryption. The front-end section of the report will cover the user interface design and layout along with the screen navigation of the application. The database section will provide an overview of the complete database and its use in this project. The main system architecture will also be detailed below. The socket section will provide an explanation for how the Java socket pair will function to support client/server communication, and the final section will describe the implementation of encryption and decryption within the application to secure the files. The main functionality and the use cases of this application have been defined in the previous documentation. The application will allow users to efficiently and securely transfer files. It will be a Windows based application and will consider ease-of-use to be a top priority.

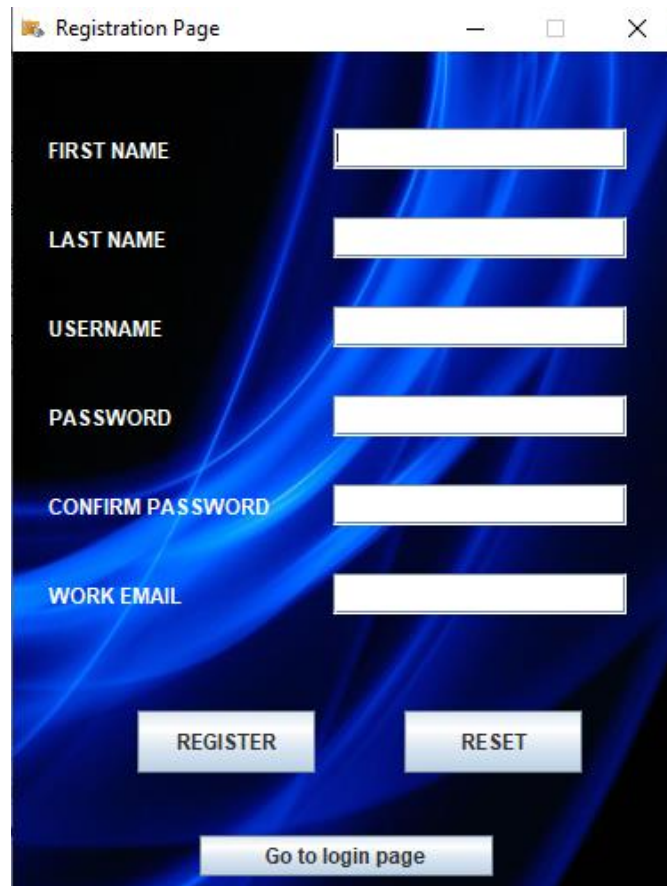
Front-end Layout

The application follows a simple navigation system where the next relevant page can be accessed from the current page. The style is used to allow users to use the application easily and intuitively. The simple user interface remains consistent throughout the application and allows users to access and send files quickly and efficiently.

Screen-flow Screenshots

1. Registration Page

Upon launching the application, users will see the registration page pictured below. Here, users will enter the information required and will become registered with the application. If a user is already registered, they can select the button at the bottom of the screen to go straight to the login page. A reset button is also provided in case a user enters incorrect credentials by mistake and wishes to start again. The reset button clears all the fields on the screen.



The image shows a software window titled "Registration Page" with standard window controls (minimize, maximize, close). The background is a dark blue abstract pattern. The form contains six input fields with labels to their left: "FIRST NAME", "LAST NAME", "USERNAME", "PASSWORD", "CONFIRM PASSWORD", and "WORK EMAIL". Below the fields are two buttons, "REGISTER" and "RESET", and a link "Go to login page" at the bottom center.

FIRST NAME	<input type="text"/>
LAST NAME	<input type="text"/>
USERNAME	<input type="text"/>
PASSWORD	<input type="password"/>
CONFIRM PASSWORD	<input type="password"/>
WORK EMAIL	<input type="text"/>

[Go to login page](#)

2. Login Page

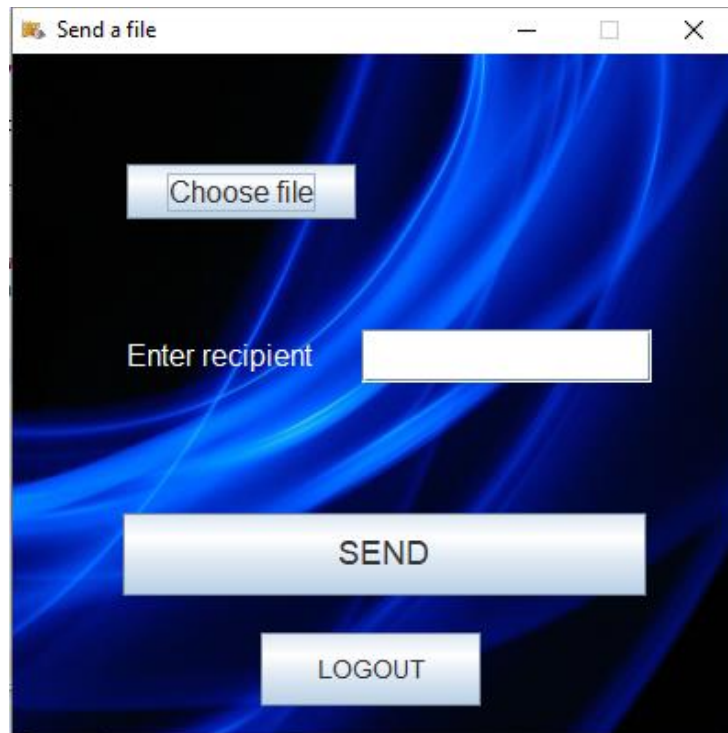
The login page looks as below. Users will enter their username and password to login to the application. A reset button is also provided in case a user enters incorrect credentials by mistake and wishes to start again. The reset button clears all the fields on the screen. Users can also click the link provided at the bottom of the screen to reset their password if necessary.



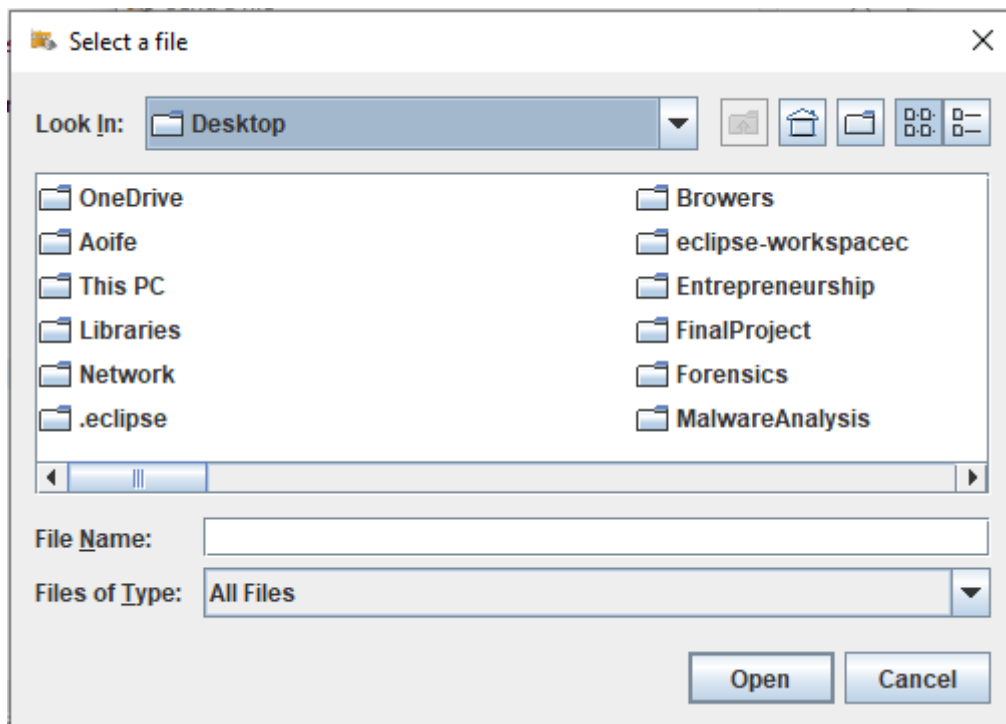
The screenshot shows a window titled "Login Page" with a blue and black abstract background. It contains two text input fields labeled "USERNAME" and "PASSWORD". Below these fields are two buttons labeled "LOGIN" and "RESET". At the bottom of the window, there is a link labeled "Forgot password".

3. Home Page

Upon users successfully logging in to the application, the home page will be automatically loaded, which will be where users send their files.

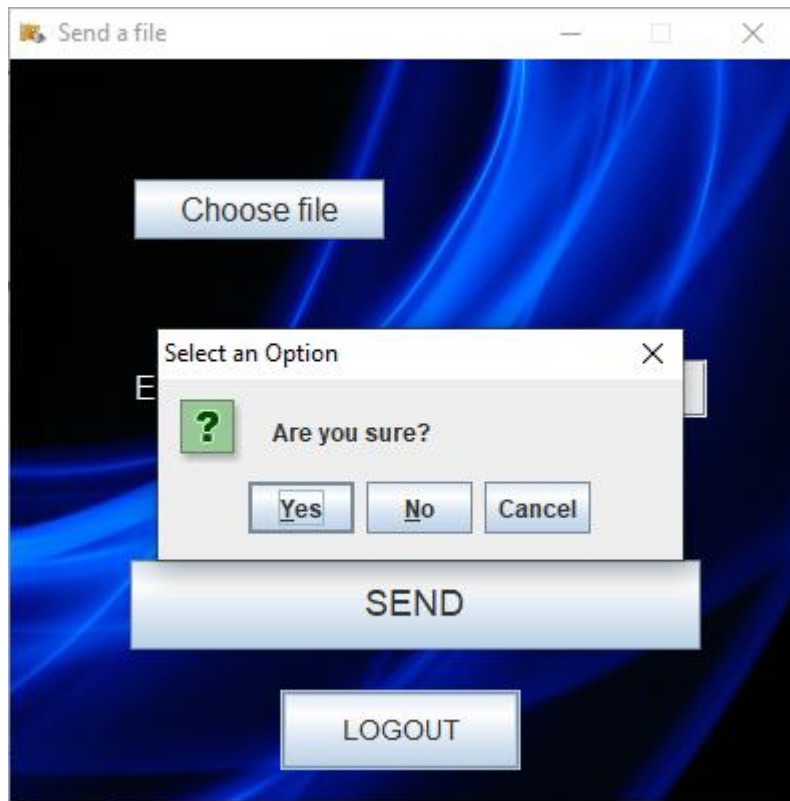


The choose file button will open a file chooser, allowing users to choose a file from their device to send using the application.



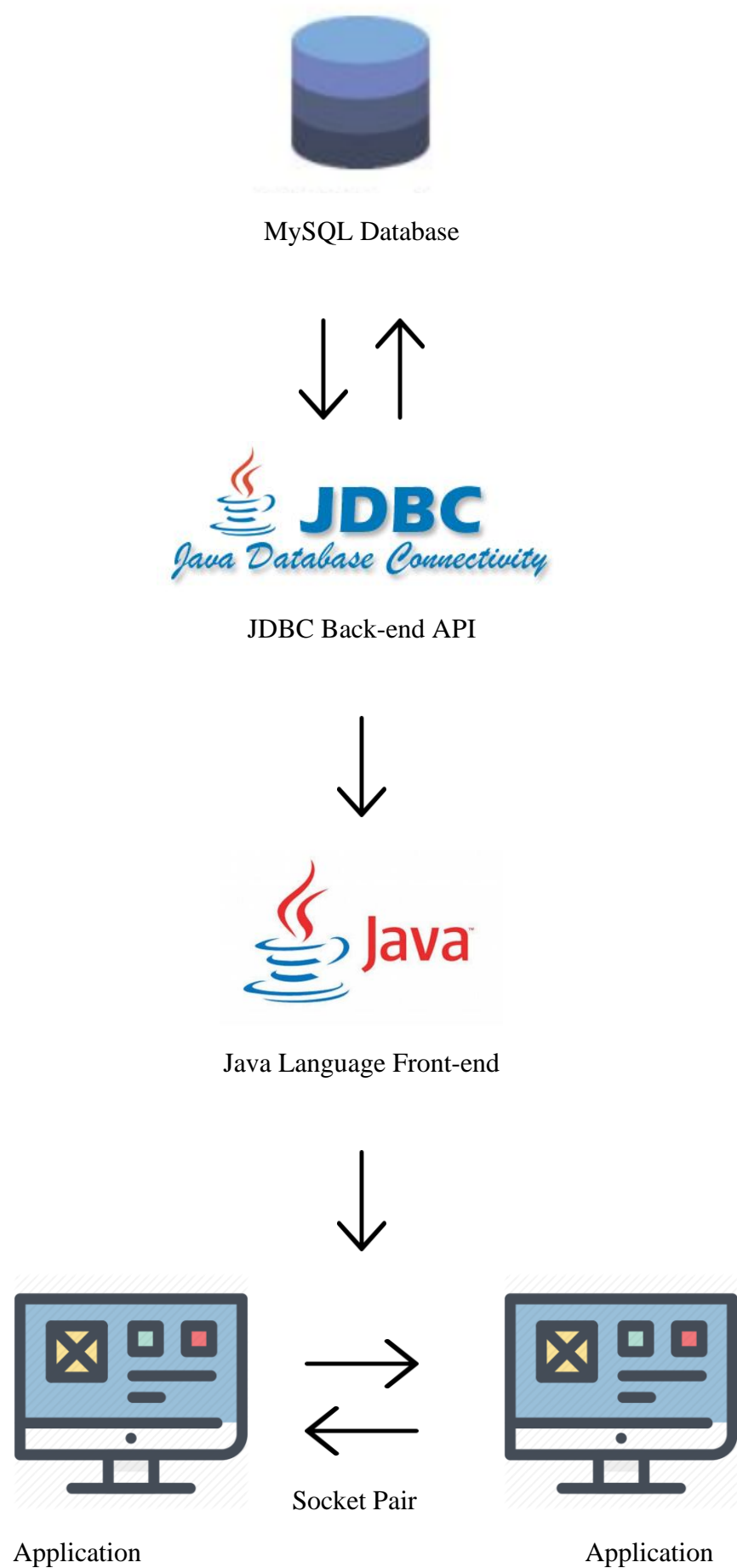
4. Logout Page

Users can logout of the application using the button provided. A pop up box will appear on the screen asking the user to confirm or cancel the logout. If a user clicks the yes option, they will be logged out successfully. If a user clicks no, they will remain logged in to the application until the choose to logout.



System Architecture

The system is comprised of a MySQL database, a Java Database Connectivity (JDBC) application programming interface (API), a Java user interface and a Java socket pair which allows for client/server communication. An API is a set of operations that can be used by the application to carry out its primary functions. The API contains functions that add, delete, update and read from the database and return that information for use by the API. The user interface uses the API by making calls and requests and receiving responses.



Database

A database is an organised collection of data used by computer applications for the storage and retrieval of information. In this application, a MySQL database will serve the purpose of storing users' information. This will be done by creating a table within the database and using the information stored there to provide the functionality required by the application.

Table Structure

Users Table

The users table contains the information about all users registered with the application. Users will be added to this table upon successful registration with the application. The table contains columns; first name, last name, username, password, email, salt, sessionID, loggedIn, public and private.

MySQL prepared statements are required for the following actions:

- Adding a new user to the table upon registration.
- Returning information about a specific user.
- Update user's information.

Field Name	Data Type	Field Description
FIRSTNAME	VARCHAR(30)	Contains the first name of the user entered at registration.
LASTNAME	VARCHAR(30)	Contains the last name of the user entered at registration.
USERNAME	VARCHAR(30)	Contains the unique username of the user entered at registration.
PASSWORD	VARCHAR(300)	Contains the salted and hashed version of the plaintext password of the user entered at registration.
EMAIL	VARCHAR(30)	Contains the email address of the user entered at registration.
SALT	VARCHAR(300)	Contains the individual random salt used with the users password.
SessionID	VARCHAR(45)	Contains a session ID for the user's session, and is generated upon a user logging in.
LoggedIn	TINYINT	Contains a variable which identifies if a user is logged in or not, i.e. 1 for yes and 0 if not.
Public	VARBINARY(4000)	Contains the symmetrically encrypted RSA public key of the user generated at registration.

Private	VARBINARY(4000)	Contains the symmetrically encrypted RSA private key of the user generated at registration.
---------	-----------------	---------------------------------------------------------------------------------------------

Adding a User (Registration Use Case):

```

PreparedStatement Pstatement=connection.prepareStatement("insert into users
values(?,?,?,?,?,?,?,?,?,?)");
Pstatement.setString(1,firstnameTextField.getText());
Pstatement.setString(2,lastnameTextField.getText());
Pstatement.setString(3,usernameTextField.getText());
Pstatement.setString(5,emailTextField.getText());
Pstatement.setNull(7,java.sql.Types.NULL);
Pstatement.setNull(8,java.sql.Types.NULL);
String password=passwordField.getText();
String confirmPassword=confirmPasswordField.getText();
if(password.equalsIgnoreCase(confirmPassword))
{
    String salt = passwordFunction.getSalt(45);
    String hashedPassword = passwordFunction.generateSecurePassword(password,
    salt);
    Pstatement.setString(6,salt);
    Pstatement.setString(4,hashedPassword);
    KeyPair keyPair = RSAEncryptionWithAES.genRSAKeys();
    KeyFactory fact = KeyFactory.getInstance("RSA");
    PublicKey publicKey = keyPair.getPublic();
    String publicK = Base64.toBase64String(publicKey.getEncoded());
    PrivateKey privateKey = keyPair.getPrivate();
    String privateK = Base64.toBase64String(privateKey.getEncoded());
    byte[] decodedKey = "*redacted*".getBytes();
    SecretKey originalKey = new SecretKeySpec(decodedKey, 0, decodedKey.length,
    "AES");
    Cipher aesCipher = Cipher.getInstance("AES");
    aesCipher.init(Cipher.ENCRYPT_MODE, originalKey);
    byte[] CipherText = aesCipher.doFinal(publicK.getBytes());
    publicK = Base64.toBase64String(CipherText);
    byte[] input =privateK.getBytes();
    byte[] CipherText2 = aesCipher.doFinal(input);
    privateK = Base64.toBase64String(CipherText2);
    Pstatement.setString(9, publicK);
    Pstatement.setString(10, privateK);
    Pstatement.executeUpdate();
    JOptionPane.showMessageDialog(null,"Registration Successful");
    frame.dispose();
    LoginPage lp = new LoginPage();
    lp.setTitle("Login");
}
else
{
    JOptionPane.showMessageDialog(null,"Passwords do not match, please
    try again");
}
} catch (Exception e1)
{
    e1.printStackTrace();
}
}

```

Return User Information (Login Use Case):

```

PreparedStatement st = (PreparedStatement) connection.prepareStatement("Select
SALT, PASSWORD from users where USERNAME=?");
PreparedStatement updateLoggedIn = (PreparedStatement)
connection.prepareStatement("Update users SET LoggedIn=true where USERNAME=?");
PreparedStatement generateSessionID = (PreparedStatement)
connection.prepareStatement("Update users SET SessionID=? where USERNAME=?");
st.setString(1, username);
updateLoggedIn.setString(1, username);
generateSessionID.setString(2, username);
ResultSet rs = st.executeQuery();
if (rs.next())
{
    salt = rs.getString("SALT");
    passwordt = rs.getString("PASSWORD");
    String hashedPassword = passwordFunction.generateSecurePassword(password,
    salt);
    if (hashedPassword.equals(passwordt))
    {
        String suuid = passwordFunction.generateSessionID();
        generateSessionID.setString(1, suuid);
        generateSessionID.executeUpdate();
        updateLoggedIn.executeUpdate();
    }

    frame.dispose();
    UserHome uh = new UserHome();
    uh.setTitle("Welcome");
}
else
{
    JOptionPane.showMessageDialog(null, "Wrong Username & Password");
}
else
{
    JOptionPane.showMessageDialog(null, "Wrong Username & Password");
}
} catch (Exception sqlException)
{
    sqlException.printStackTrace();
}
}

```

Update User Information (Logout Use Case):

```

PreparedStatement getSessionID = (PreparedStatement)
connection.prepareStatement("Select SessionID from users where USERNAME=?");
PreparedStatement updateLoggedIn = (PreparedStatement)
connection.prepareStatement("Update users SET LoggedIn=false where USERNAME=?");
PreparedStatement updateSessionID = (PreparedStatement)
connection.prepareStatement("Update users SET SessionID=null where USERNAME=?");
getSessionID.setString(1, username);
updateLoggedIn.setString(1, username);
updateSessionID.setString(1, username);
ResultSet rs = getSessionID.executeQuery();

```

```

updateLoggedIn.executeUpdate();
updateSessionID.executeUpdate();
int a = JOptionPane.showConfirmDialog(null, "Are you sure?");
if (a == JOptionPane.YES_OPTION)
{
    frame.dispose();
    LoginPage frame = new LoginPage();
    LoginPage.clearScreen();
    JOptionPane.showMessageDialog(null, "You are now logged out. Please log in
to continue!");
}
else
{
}
} catch (SQLException sqlException)
{
    sqlException.printStackTrace();
}
}

```

Sockets

“A socket is one end-point of a two-way communication link between two programs running on the network. Socket classes are used to represent the connection between a client program and a server program. TCP provides a reliable, point-to-point communication channel that client-server applications on the Internet use to communicate with each other. To communicate over TCP, a client program and a server program establish a connection to one another. Each program binds a socket to its end of the connection. To communicate, the client and the server each reads from and writes to the socket bound to the connection. (Oracle 2019).

A server socket and a client socket pair will be used to allow for communication and the transfer of the files between users of the application, as pictured in the architecture diagram above.

Encryption and Decryption

As stated in the functional specification for this application, a hybrid encryption approach will be used to protect the data.

“Hybrid encryption is a mode that merges two or more encryption. It incorporates a combination of asymmetric and symmetric encryption to benefit from the strengths of each form of encryption. These strengths are respectively defined as speed and security.

Hybrid encryption is considered a highly secure type of encryption as long as the public and private keys are fully secure” (Techopedia 2012).

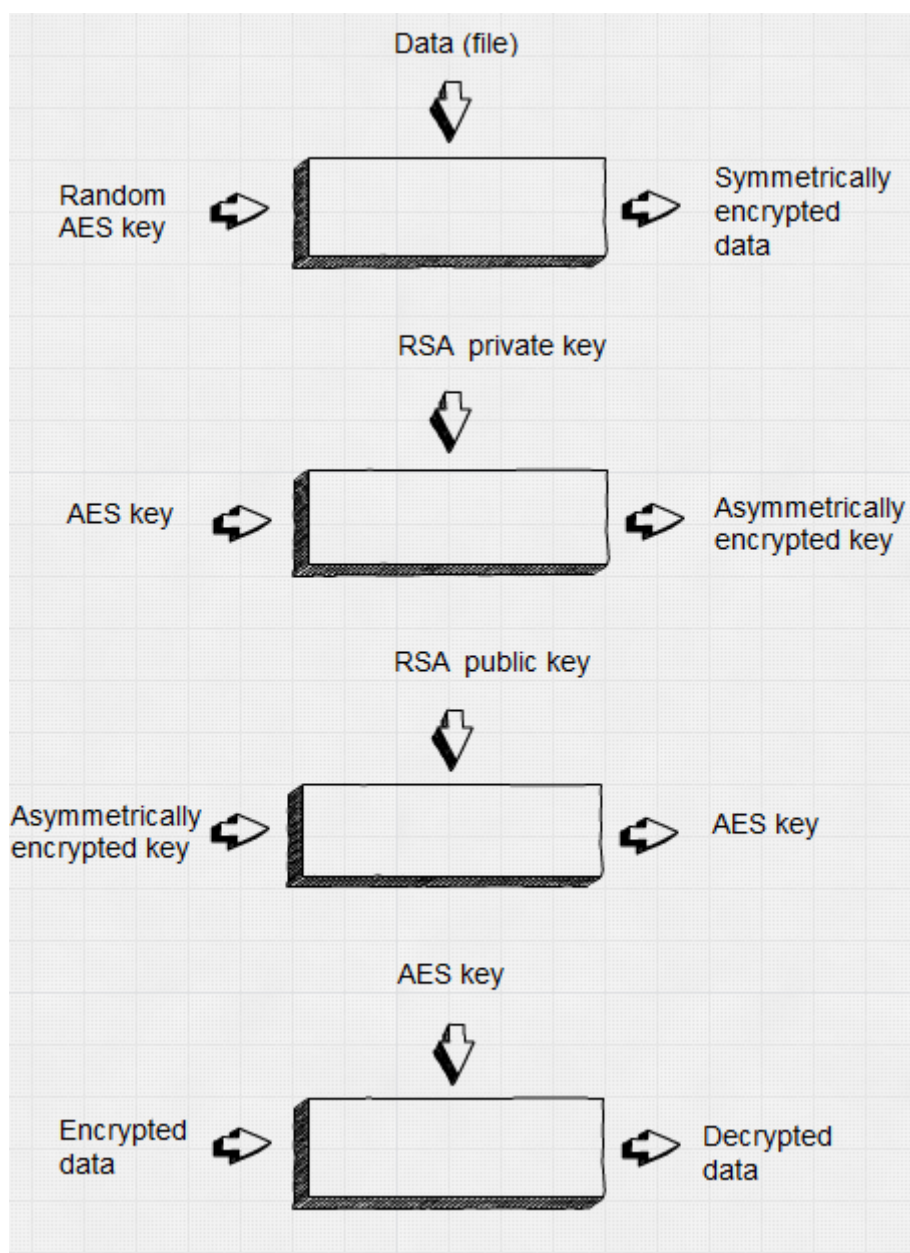
The below diagram portrays how hybrid encryption will be used in this application:

Firstly, the file will be encrypted using the symmetric encryption algorithm known as AES (The Advanced Encryption Standard).

Next, the AES (symmetric) key will be encrypted using the RSA (asymmetric) private key of the recipient.

On the other side, the recipients RSA (asymmetric) public key will be used to decrypt the asymmetrically encrypted key.

The decrypted original AES key will then be used to decrypt the file itself.



(Created using SimpleDiagrams.com, 2020)

References

Techopedia.com, 2012. *Definition – What does Hybrid Encryption mean?* [Online]
Available at: < <https://www.techopedia.com/definition/1779/hybrid-encryption> > [Accessed 20th
April 2020].

Oracle.com, 2019. *Lesson: All about Sockets* [Online] Available at:
<<https://docs.oracle.com/javase/tutorial/networking/sockets/index.html>> [Accessed 16th April
2020].