Security of Computer Systems

Project Report

Authors:
Piotr Zarzycki 180542

Version: 2.0

Wersje

Wersja	Data	Opis zmian
1.0	28.04.2022	Stworzenie dokunentu
2.0	14.06.2022	Ukończenie projektu

1. Projekt – termin kontrolny

1.1 Opis

Głównym celem tego projektu jest stworzenie aplikacji pozwalającej na bezpieczne przesyłanie wiadomości i plików za pomocą Ethernetu. Ma ona realizować możliwość szyfrowania wiadomości za pomocą szyfru ECB lub CBC. Aplikację zrealizowałem z wykorzystaniem języka java. Do przesyłu wiadomości zastosowałem sockety i server sockety.

1.2 Aktualnie zrealizowane zadania

Aktualnie została zaimplementowana komunikacja z wykorzystaniem szyfrowania ECB.

```
listening on port: 64858
Z kim chcesz się połączyć?
64864
```

Po podaniu portu, na którym znajduje się nasz rozmówca (użytkownik B) zostaje nawiązane połączenie TCP z nim. Po otrzymaniu tego połączenia użytkownik B wysyła jawnie swój klucz publiczny do użytkownika A.

```
listening on port: 64864
Z kim chcesz się połączyć?
Wyslano klucz publiczny.
```

Następnie po otrzymaniu klucza publicznego użytkownika B, użytkownik A może wpisać wiadomość, którą pragnie wysłać. Wiadomość ta zostanie zaszyfrowana przy użyciu szyfrowania ECB z użyciem otrzymanego klucza publicznego, a następnie wysłana.

```
Otrzymano klucz publiczny:
MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQDt4y+w+qNDpRSPRNJFQOH9obXqgOCjHiiM/itTZC8OyHoeCiZ9
Wpisz zawartość wiadomości:
jest dobrze
```

O poprawnym wysłaniu wiadomości, użytkownika A zostanie poinformowany komunikatem, a następnie będzie mógł wysyłać kolejne wiadomości do kolejnych użytkowników.

```
Wysłano wiadomość.
```

Użytkownik B dostanie natomiast zaszyfrowaną wiadomość, którą odszyfruje przy pomocy swojego klucza prywatnego i wyświetli ją użytkownikowi.

```
U+qKspcyUxqYHqGVDnraXB+XpJHaQIP2VJr+pJocJbxhoTSFcqDg2J9KL++vBUMYyB7uXMoPXVU9xczVYqlhNj7FF3fGPXuoL
Po odszyfrowaniu:
jest dobrze
```

Użytkownicy mogą w jednej chwili odbierać wiadomości od wielu nadawców oraz jednocześnie wysyłać własne wiadomości dzięki zastosowaniu wielowątkowości.

1.3 *Implementacja*

Klasa Client zawiera w sobie część odpowiedzialną za wysyłanie wiadomości oraz obsługę żądań użytkownika.

```
ObjectOutputStream output = new ObjectOutputStream(socket.getOutputStream());
ObjectInputStream input = new ObjectInputStream(socket.getInputStream())){

String publicKey = (String) input.readObject();
System.out.println("Otrzymano klucz publiczny: ");
System.out.println(publicKey);

System.out.println("Wpisz zawartość wiadomości:");
String message = scan.nextLine();
try{
    String encryptedMessage = rsa.encryptECB(message, publicKey);
    output.writeObject(encryptedMessage);
    output.flush();
    System.out.println("Wysłano wiadomość.");
```

Jest także odpowiedzialna za wystartowanie wątku odpowiedzialnego za obsługiwanie wiadomości które przychodzą do użytkownika.

```
RSA rsa = new RSA();

ServerSocket ss = new ServerSocket( port: 0);
Thread reciveMessageThread = new Thread(new ReciveMessage(ss, rsa));
reciveMessageThread.start();
System.out.println("listening on port: " + ss.getLocalPort());
```

Klasa ReciveMessage służy jedynie do delegowania nowych połączeń od użytkowników do nowych wątków, aby umożliwić obsługiwanie wielu przychodzących wiadomości jednocześnie.

```
while (true){
    Socket s = null;
    try {
        s = ss.accept();
        for (Thread thread : threads) {
            if (!thread.isAlive()) {
                threads.remove(thread);
            }
        }

        Thread t = new Thread(new ReceiveMessageHandler(s, rsa));
        threads.add(t);
        t.start();
    }
    catch (IOException ignored){}
}
```

Klasa ReciveMessageHangler służy do obsługi poszczegulnych połączeń. Wysyła klucz publiczny użytkownika i oczekuje na wiadomość aby ją następnie odkodować i wyświetlić użytkownikowi.

```
(ObjectOutputStream output = new ObjectOutputStream(socket.getOutputStream());
 ObjectInputStream input = new ObjectInputStream(socket.getInputStream())) {


output.writeObject(Base64.getEncoder().encodeToString(rsa.publicKey.getEncoded()));
System.out.println("Wysłano klucz publiczny.");

String encryptedMessage = (String) input.readObject();
System.out.println("Otrzymano wiadomość:");
System.out.println(encryptedMessage);
System.out.println("Po odszyfrowaniu:");
System.out.println(rsa.decryptECB(encryptedMessage));
```

Klasa RSA natomiast zawiera implementację szyfrowania RSA ECB oraz przechowuje klucz publiczny i prywatny.

```
private PrivateKey privateKey;
public PublicKey publicKey;

public RSA() {
    try {
        KeyPairGenerator generator = KeyPairGenerator.getInstance("RSA");
        generator.initialize( keysize: 1024);
        KeyPair pair = generator.generateKeyPair();
        privateKey = pair.getPrivate();
        publicKey = pair.getPublic();
    } catch (Exception ignored) {}
}
```

```java
private PublicKey publicKeyFromString(String publicKeyString){
    PublicKey publicKey = null;
    try{
        X509EncodedKeySpec keySpecPublic = new X509EncodedKeySpec(decode(publicKeyString));

        KeyFactory keyFactory = KeyFactory.getInstance("RSA");

        publicKey = keyFactory.generatePublic(keySpecPublic);
    }catch (Exception ignored){}
    return publicKey;
}

private String encode(byte[] data){
    return Base64.getEncoder().encodeToString(data);
}

public String encryptECB(String message, String publicKeyString) throws Exception{
    byte[] messageToBytes = message.getBytes();
    Cipher cipher = Cipher.getInstance("RSA/ECB/PKCS1Padding");
    PublicKey publicKey = publicKeyFromString(publicKeyString);
    cipher.init(Cipher.ENCRYPT_MODE,publicKey);
    byte[] encryptedBytes = cipher.doFinal(messageToBytes);
    return encode(encryptedBytes);
}
```

```java
private byte[] decode(String data){
    return Base64.getDecoder().decode(data);
}

public String decryptECB(String encryptedMessage) throws Exception{
    byte[] encryptedBytes = decode(encryptedMessage);
    Cipher cipher = Cipher.getInstance("RSA/ECB/PKCS1Padding");
    cipher.init(Cipher.DECRYPT_MODE,privateKey);
    byte[] decryptedMessage = cipher.doFinal(encryptedBytes);
    return new String(decryptedMessage, charsetName: "UTF8");
}
```

1.4 Plany rozszerzenia

- Dodanie szyfrowania CBC
- Generowanie klucza sesyjnego
- Zapisywanie klucza publicznego oraz prywatnego RSA na dysku użytkownika i zaszyfrowanie go.
- Implementacja GUI
- Dodanie możliwości przesyłania plików
- Dodanie możliwości przesyłania dużych plików oraz pasek postępu przedstawiający progres przesyłania
- Przeprowadzenie testów - stworzenie testów automatycznych

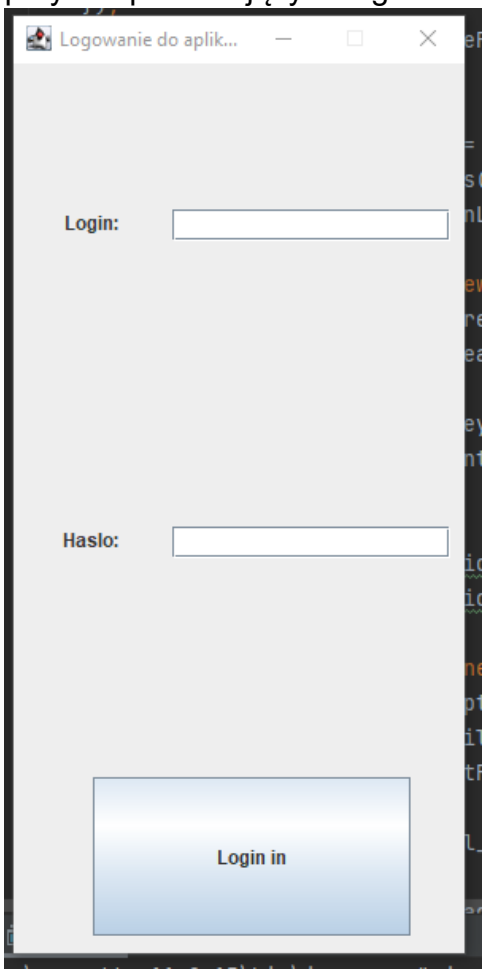
2. Projekt - termin końcowy

2.1 Opis

Głównym celem tego projektu pozostał niezmieniony. Dalej jest nim stworzenie aplikacji pozwalającej na bezpieczne przesyłanie wiadomości i plików za pomocą Ethernetu. Ten bezpieczny przesył informacji został zrealizowany za pomocą szyfrów ECB lub CBC w trybie AES. Jako że są to sposoby szyfrowania są sposobami szyfrowania symetrycznego potrzebne było także zastosowanie szyfrowania asymetrycznego RSA/ECB. Było ono wykorzystywane na początku każdej komunikacji, aby ustalić klucz sesyjny wykorzystywany podczas reszty wymiany. Aplikację została z wykorzystaniem języka java, a do przesyłu wiadomości zastosowałem sockety i server sockety.

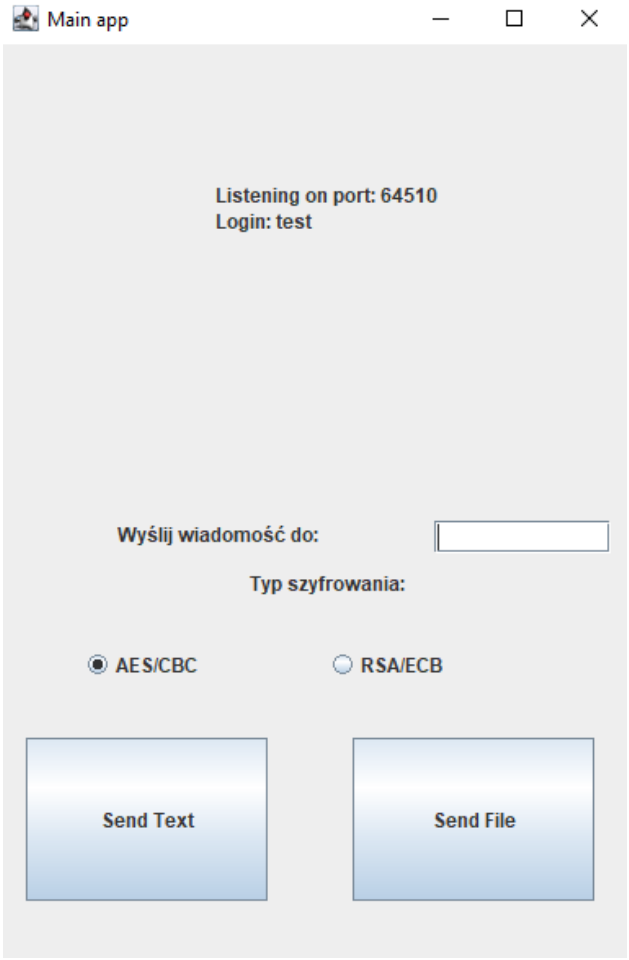
2.2 Projekt GUI

Przy każdym odpaleniu aplikacji pierwszym ekranem, który zostanie nam zaprezentowany jest okno logowania zawierające pola loginu i hasła a także przycisk pozwalający zalogować się do aplikacji.



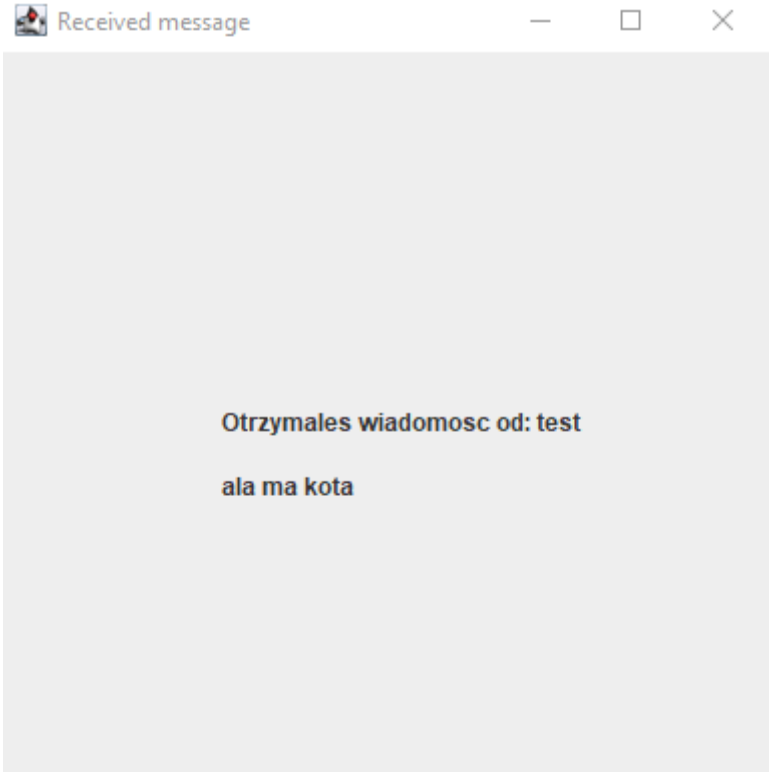
Po zalogowaniu się zostaniemy przywitani głównym oknem aplikacji informującym nas jako jaki użytkownik jesteśmy zalogowani, na jakim porcie nasłuchujemy i oczekujemy wiadomości z innych źródeł. Oprócz tych informacji mamy także pola pozwalające nam wysłać wiadomości do innych użytkowników. Pole tekstowe pozwala nam na wpisanie nr portu osoby, której chcemy wysłać wiadomość. Znajdują się także 2 raido buttony pozwalające na wybór 1 z 2 typów

szyfrowania. Pod nimi znajdują się 2 przyciski pozwalające na wybór jakiego rodzaju wiadomość chcemy wysłać.



Po wybraniu możliwości wysłania wiadomości tekstowej otwiera nam się okno z polem na wpisanie wiadomości i przyciskiem pozwalającym ją wysłać.

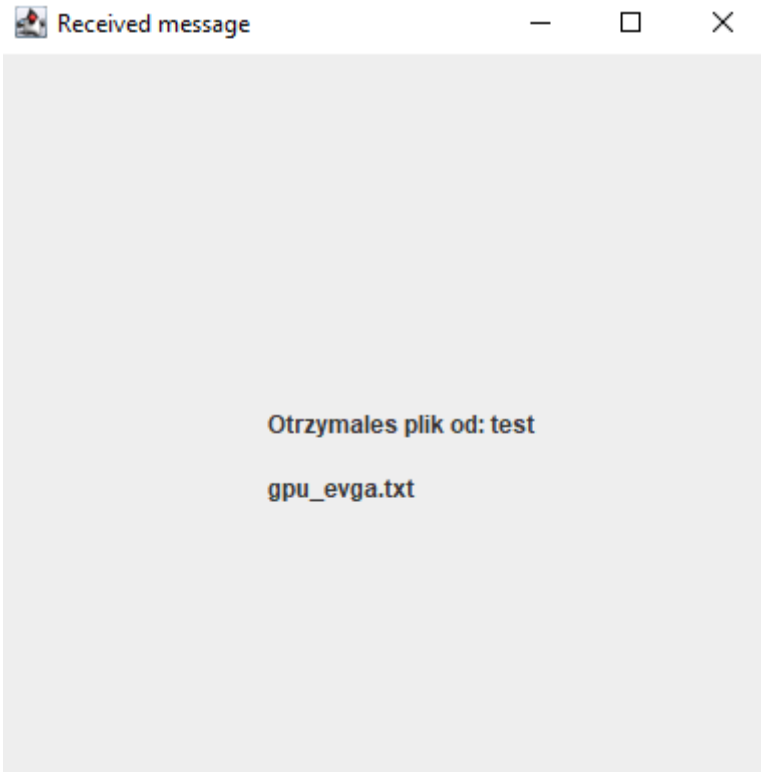
Po wysłaniu tej wiadomości drugi użytkownik otrzyma wiadomość w nowym okienku wraz z loginem osoby która ją wysłała.



W wypadku kiedy z głównego okna aplikacji naciśniemy dla odmiany przycisk

wysłania pliku zostanie wyświetlone odpowiednie okienko, które zamiast pola na wpisanie tekstu będzie posiadało przycisk pozwalający na wybór pliku.

Po odebraniu pliku przez drugą stronę zostanie wyświetlona informacja w okienku podobnym do tego którego dostajemy po otrzymaniu wiadomości tekstowej, o osobie, która wysłała nam ten plik jak i jego nazwa. Pliki te zapisują się w folderze aplikacji.



2.3 *Komunikaja*

Tak jak zostało to już wcześniej wspomniane do komunikacji między aplikacjami zostały wykorzystane sockety i server sockety javowe które wykożystują protokuł TCP.

```
try (Socket socket = new Socket( host: "localhost", destination)) {
    try(ObjectOutputStream output = new ObjectOutputStream(socket.getOutputStream());
        ObjectInputStream input = new ObjectInputStream(socket.getInputStream())){
```

Aplikacja frontendowa pozwalająca wysyłać wiadomości i aplikacja pozwalająca na odbieranie nadchodzących wiadomości działają na różnych wątkach co pozwala na jednoczesne wysyłanie i odbieranie wiadomości. Dodatkowo do odbierania wiadomości wykorzystano system delegujący każde nadchodzące

połączenie do nowego wątku.

```
while (true){
    Socket s = null;
    try {
        s = ss.accept();
        for (Thread thread : threads) {
            if (!thread.isAlive()) {
                threads.remove(thread);
            }
        }

        Thread t = new Thread(new ReciveMessageHandler(s, rsa));
        threads.add(t);
        t.start();
    }
    catch (IOException ignored){}
}
```

2.4 *Wymiana kluczy*

W momencie odebrania chęci nawiązania połączenia (osoba A chce nam wysłać wiadomość) wątek odbierający wiadomości (osoby B) wysyła osobie A jawnie swój klucz publiczny

```
output.writeObject(rsa.encode(rsa.publicKey.getEncoded()));
System.out.println("Wysłano klucz publiczny.");
```

Aplikacja użytkownika A tworzy klucz sesyjny i wektor inicjujący i zapisuje je w zmiennej klasy HeaderMessage. Zmienna ta oprócz tych 2 wartości posiada także informację, ile wiadomości zostanie wysłanych (w wypadku plików na ile trzeba było je podzielić) oraz informację o stosowanym typie szyfrowania. Następnie szyfruje te informacje z wykorzystaniem klucza publicznego osoby B i szyfrowania RSA/ECB/PKCS1Padding. Po zaszyfrowaniu ich wysyła je użytkownikowi B.

```
String sesionKey = AES.generateKeyString();
String sesionIv = AES.generateIvString();

HeaderMessage messageHeader = HeaderMessage.builder() Header
        .encryptionType(decidedEncryption) capture of ?
        .sesionKey(sesionKey)
        .initVector(sesionIv)
        .build();

messageHeader.encrypt(rsa, publicKeyPartner);
output.writeObject(messageHeader);
output.flush();
System.out.println("Wysłano header");

AES aes = new AES(sesionKey, decidedEncryption, sesionIv);
```

Użytkownik B otrzymuje zaszyfrowaną wiadomość z informacjami potrzebnymi do szyfrowania przyszłych wiadomości podczas tej wymiany i odszyfrowuje go za pomocą swojego klucza prywatnego.

```
HeaderMessage headerMessage = (HeaderMessage) input.readObject();
headerMessage.decrypt(rsa);
AES aes = new AES(headerMessage.sesionKey,headerMessage.encryptionType,headerMessage.initVector);
```

2.5 *Wysyłanie wiadomości*

Na początku cała wiadomość jest szyfrowana przy pomocy szyfru i klucza sesyjnego ustalonego podczas wymiany. Następnie wiadomości plikowe są wysyłane poprzez podział ich na mniejsze fragmenty 16384 bitowe. Zostają zapisane w strukturze FileMessage która zawiera w sobie informacje o osobie która wysyła tą wiadomość, oryginalną nazwę pliku który wysyłamy, części zawartości wiadomości oraz numer która to część pozwalający na ponowne przyszłe odtworzenie tej wiadomości.

```
FileInputStream inputStream = new FileInputStream(encryptedFile);
byte[] buffer = new byte[16384];
int bytesRead;
int nrOfMessage = 0;
while ((bytesRead = inputStream.read(buffer)) != -1) {
    FileMessage encryptedMessage = new FileMessage();
    encryptedMessage.setOwner(login);
    encryptedMessage.setContent(buffer);
    encryptedMessage.setNumberOfPart(nrOfMessage);
    encryptedMessage.setFileName(file.getName());

    nrOfMessage++;

    encryptedMessage.encrypt(aes);

    output.writeObject(encryptedMessage);
    output.flush();
    System.out.println("Wysłano wiadomosc" + nrOfMessage);
```

Użytkownik B odbiera te wiadomości w z góry nie kolejności ponieważ jest wykorzystywane przesyłanie poprzez strumień wykorzystujący protokół TCP. Dlatego użytkownik dekoduje tylko informacje o nadawcy nazwie pliku i nr częsci a samej zawartości pliku nie deszyfruje do momentu otrzymania ich wszystkich.

```
AES aes = new AES(headerMessage.sesionKey,headerMessage.encryptionType,headerMessage.initVector);

Message message = null;
ArrayList<Message> messages = new ArrayList<>();
for (int i = 0; i < headerMessage.getNumberOfParts(); i++) {
    message = (Message) input.readObject();
    message.decrypt(aes);
    messages.add(message);

    System.out.println("Odebrano wiadomosc " + (i + 1));
}

message.merge(messages, aes);
message.show();
```

Po odebraniu wszystkich składowych pliku wykonywane sortowanie ich a następnie łączenie ich w jeden zaszyfrowany plik tymczasowy.

```
File encryptedFile = new File( pathname: "tempFiles\\encryptedFile" + getFileName());
try {
    encryptedFile.createNewFile();
    FileOutputStream outputStream = new FileOutputStream( name: "tempFiles\\encryptedFile" + getFileName());
    messages.sort((a , b) -> {
        return a.getNumberOfPart() - b.getNumberOfPart();
    });
    for (Message<byte[]>message : messages) {
        if (message.getContent() != null) {
            outputStream.write(message.getContent());
        }
    }
    outputStream.close();
} catch (IOException e) {
    e.printStackTrace();
}
```

Następnie plik ten jest deszyfrowany z wykorzystaniem klucza sesyjnego i zapisany w docelowym miejscu.

```
File decryptedFile = new File( pathname: "ReceivedFiles\\" + getFileName());
try {
    decryptedFile.createNewFile();
    aes.decryptFile(encryptedFile, decryptedFile);
    encryptedFile.delete();
} catch (Exception e) {
    e.printStackTrace();
}

this.setOwner(messages.get(0).getOwner());
```

Odbieranie wiadomości tekstowych wygląda dokładnie tak samo dzięki zastosowaniu polimorfizmu. Dla nich funkcja merge łączy zawartości wszystkich wiadomości w jeden string.
Dzięki zastosowaniu podziału na mniejsze wiadomości i ponownemu składaniu w jedną można bezproblemowo przesyłać wiadomości niezależnie od ich rozmiaru.

2.6 *Zapisywanie kluczy asymetrycznych*

W momencie, kiedy tworzony jest użytkownik z nowym, jeszcze niespotykanym loginem tworzona jest nowa para kluczy prywatnego i publicznego i zapisywane są w pliku. Dodatkowo klucz prywatny przed zapisaniem w pliku jest szyfrowany z wykorzystaniem szyfrowania AES/CBC/PKCS5Padding, wektora inicjalizującego “AAAAAAAAAAAAAAAAAAAAAA==” oraz skrótu klucza z wykorzystaniem funkcji hashującej MD5.

```
private void saveKeys(String login, String password){
    String passwordHash = getHash(decode(password));
    String privateKeyEncrypted = null;
    try {
        AES aes = new AES(passwordHash);
        privateKeyEncrypted = aes.encrypt(encode(privateKey.getEncoded()));
    } catch (Exception e) {
        System.out.println("Failed to encrypt privateKey");
    }

    saveKey(location(login, isPrivate: true),privateKeyEncrypted);
    saveKey(location(login, isPrivate: false),encode(publicKey.getEncoded()));
```

```
public static String getHash(byte[] input) {
    try {
        MessageDigest messageDigest = MessageDigest.getInstance("MD5");
        messageDigest.reset();
        messageDigest.update(input);
        byte[] digestedBytes = messageDigest.digest();
        String hashValue = encode(digestedBytes);
        return hashValue;
    } catch (NoSuchAlgorithmException e) {
        System.out.println("Hash function is not working");
        return "";
    }
}
```

W wypadku, kiedy użytkownik z podanym loginem już istnieje wykonywana jest funkcja odzyskiwania kluczy ponownie wykożystując ten sam zestaw parametrów co w wypadku ich szyfrowania.

```
String privateKeyEncrypted = retriveKey(location(login, isPrivate: true));
String publicKeyDecrypted = retriveKey(location(login, isPrivate: false));

String passwordHash = getHash(decode(password));
AES aes = new AES(passwordHash);

String privateKeyDecrypted = privateKeyEncrypted;
try {
    privateKeyDecrypted = aes.decrypt(privateKeyEncrypted);
} catch (Exception e) {
    System.out.println("Failed to decrypt privateKey");
}

X509EncodedKeySpec keySpecPublic = new X509EncodedKeySpec(decode(publicKeyDecrypted));
PKCS8EncodedKeySpec keySpecPrivate = new PKCS8EncodedKeySpec(decode(privateKeyDecrypted));

KeyFactory keyFactory = KeyFactory.getInstance("RSA");

publicKey = keyFactory.generatePublic(keySpecPublic);
privateKey = keyFactory.generatePrivate(keySpecPrivate);
```

W wypadku podania złego hasła nie uda się odszyfrować klucza prywatnego co oznacza, że nie będziemy w stanie odbierać wiadomości wysyłanych do nas. Dokładniej ujmując wiadomości będą wysyłane, ale podczas próby ich odczytania poleci exception i wątek odbierający wiadomości zakończy swoje działanie.

2.7 Podsumowanie

Aplikacja pozwala na bezpieczne przesyłanie wiadomości i plików. Jest zaimplementowana w sposób, który nie ogranicza funkcjonalności jej podczas otrzymywania licznych wiadomości. Część aplikacji odpowiedzialna za otrzymywanie wiadomości jest zaimplementowana w sposób uniwersalny pozwalająca na jednakowe obchodzenie się z nadchodzącymi wiadomościami nie zależnie od ich typu. Przesyłanie plików nie jest w żaden sposób ograniczone, jeżeli chodzi od dostępne rozszerzenia. Dzięki dzieleniu dużych wiadomości na liczne mniejsze możliwe jest przesyłanie wiadomości o dowolnym rozmiarze. Implementacja rodzajów szyfrowania także pozwala na łatwe rozszerzenie o pozostałe z szyfrów z rodziny AES.

3. Literatura

[1] https://www.thexcoders.net/rsa-for-client-server/.

[2] https://www.geeksforgeeks.org/asymmetric-encryption-cryptography-in-java/.

[3] https://stackoverflow.com/

[4] https://www.baeldung.com/java-aes-encryption-decryption

[5] https://howtodoinjava.com/java/java-security/java-aes-encryption-example/