Programming for Security

Spring Semester 2021

Project Assignment

Cansu Moran

Description of the Application:
In this project, a password manager is implemented. In order to login to the password manager, the user needs to enter a password when the program is run for the first time. The password entered for the first time becomes the master password to login to the application. After logging in with the first password, the user cannot change the password, unless the encrypted file containing the passwords is deleted. Once logged in, the user can store the URL, username and password information in the application. All the information is saved and stored as an encrypted file once the user closes the application. When the user is logged in and the program is still running, the encrypted file is decrypted and the decrypted file exists throughout the execution of the program. Therefore, it is essential that the user closes the application once they are done, to make sure the decrypted file is deleted and no one else has access to the stored passwords. In addition, since in the program the user is asked to enter the password only during login, once the user is logged in, all the stored passwords are visible on the application. Therefore, the user needs to log off to make sure someone else using the computer after them don't see the passwords.

Security Goals:
The idea behind the password manager is to keep all the passwords of the user in one place, and allowing them to access all the stored passwords using one master key. The users tend to use passwords that are easy to remember, which may include their name or a specific date. It is also likely that they reuse these simple passwords for different websites. The usage of such short and reused passwords create a risk of their passwords being cracked easily and if someone has access to one password, they can log into multiple websites as that user. To prevent these security problems, a password manager can be used. A password manager helps the user to store all their passwords in one application and have access to them by simply remembering one master key that gives them access to stored data in the application. This way the user can use long and hard to crack passwords that are different for each website. In the development of the password manager, the main goal is to keep the passwords confidential with only one way to access them, which is the main password. To achieve this goal, the password information is encrypted in a file using CBC, where the IV is generated randomly and the encryption key for the file is derived from the master password. In terms of integrity, a password manager shouldn't allow anyone with an access to the computer, where the encrypted password information is stored, to tamper with the passwords, or if tampered, let the user know about it. However, this was not implemented due to time constraints. In addition, the password manager should prevent brute forcing the main password. This can be achieved by adding another layer of security such as CAPTCHA to ensure the user is a person or by only letting them enter the password incorrectly a certain number of times and blocking them for some time after consecutive wrong trials. This feature was also not implemented in this project.

Threats to the Application:
- The passwords are stored inside an encrypted file in the user's computer. If someone else deletes the file, all the passwords will be gone. In addition, if someone tampers with the file (e.g. change the IV in the file name, change the file type from "aes" to something else) the passwords and the password retrieval process can be damaged.
- Once the user is logged in, all passwords are visible. The user needs to remember to close the application, otherwise once logged in, anyone using the computer can see all the information. In addition, when the user is logged in and the program is running, a decrypted file containing all the passwords is created. The user needs to close the application for that file to be deleted and all information to be encrypted again.
- Since there is only one master password, if someone finds out what the password is, they will have access to all the other passwords.
- When the user is entering master password for the first time, there is no limitation to how secure the password can be. The only restriction is that they cannot enter an empty password. However, since there is no check to see how secure the password is, the application is as secure as the password the user uses to log in. If they use a simple password, since there is no other security check (e.g. CAPTCHA control, blocking after consecutive wrong trials), the password can be cracked by someone else using brute force.

Design Pattern:
In the program, single access point design pattern is used. This design pattern is chosen because it allows a simple and intuitive way to enter the application for the user. Once they are logged in, they don't need to enter the password again, which makes it easier for them to use. Although this design choice makes the usage of the application easier for the user, it also creates certain threats to the security of the program. Once the user logs in, the user has access to all the websites, usernames, and passwords. If the user leaves the program running and someone else uses the computer, they will have access to all the passwords of the user as well, since the initial point of access has already been passed. In

addition, having one password protecting all the other passwords can be dangerous if the user uses a simple password. Since there is only one login with one password, someone can brute force possible password combinations to gain access to the information.

Test of the Application:
In the test case, the decrypt method is tested. The decrypt method located in Security class is used to decrypt an already encrypted file located in “files” directory. The method returns true if the file can be decrypted and false otherwise. In order to test this method, two tests are done. One with right password to see if the decryption is successful and one with wrong password to see if it actually fails. Since the decrypt methods requires that there is already an encrypted file in the directory, encrypt method is called before each decrypt test. In addition, after each test, the directory needs to be cleaned since there will be decrypted files that can tamper with the execution of the actual program. Since after the test, the all the files in the “files” directory will be deleted, the tests should not be done after executing the program, since the information coming from the actual program will also be deleted with the files generated by the tests. The tests for the decrypt method are run and they both have been passed. In other words, the decrypt method can successfully decrypt an encrypted file with the right password and return true and if the password is incorrect, it returns false.

Building and Running the Program:
In the program, the main class is named as Main.java. The program uses the same versions of Bouncy Castle and JavaFx as listed in the Moodle page. In addition to these libraries, version 1.1.1 of Json Simple library is used for the program and version 2.20.0 of Mockito JUnit Jupiter library is used for the tests. The links for these libraries are given below:
Mockito JUnit Jupiter 2.20.0 : https://dl.bintray.com/mockito/maven/org/mockito/mockito-junit-jupiter/2.20.0/mockito-junit-jupiter-2.20.0.jar
Json Simple 1.1.1 : https://storage.googleapis.com/google-code-archive-downloads/v2/code.google.com/json-simple/json-simple-1.1.1.jar

UML Class Diagram:

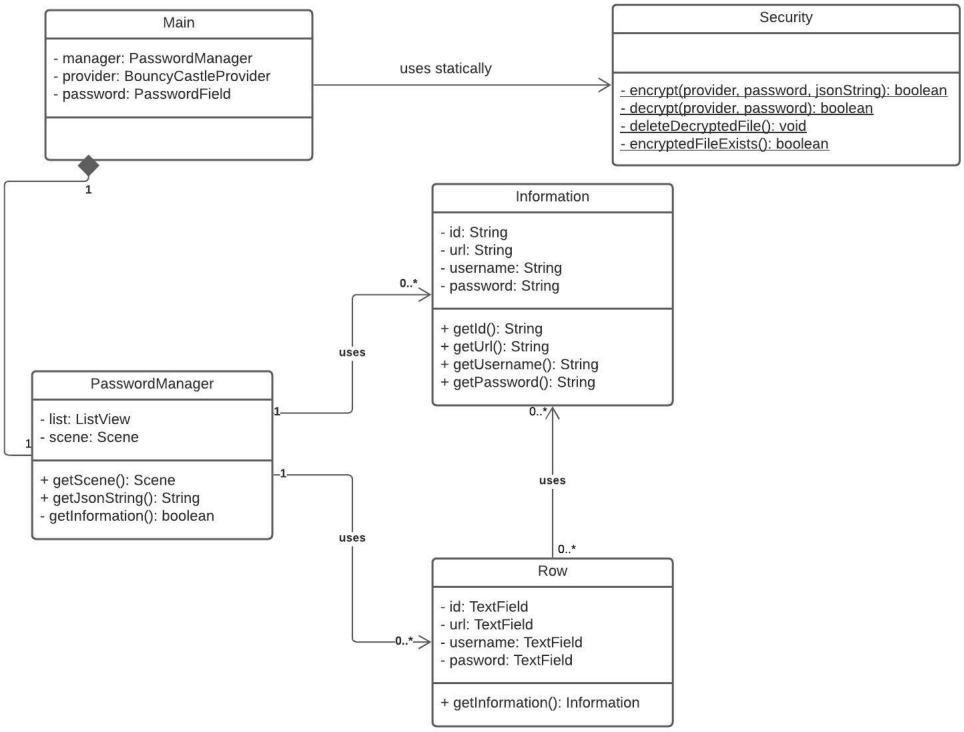

Figure 1: UML Class Diagram

Screenshots from the Application:

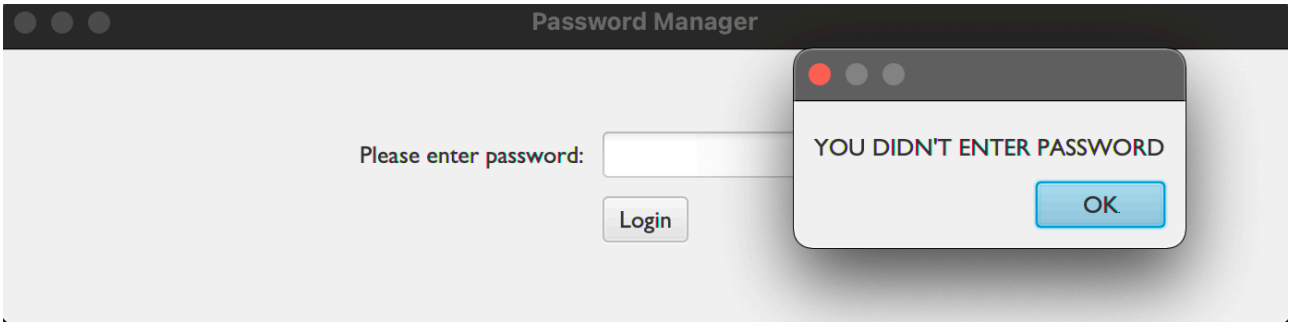

Figure 2: Trying to enter with an empty password

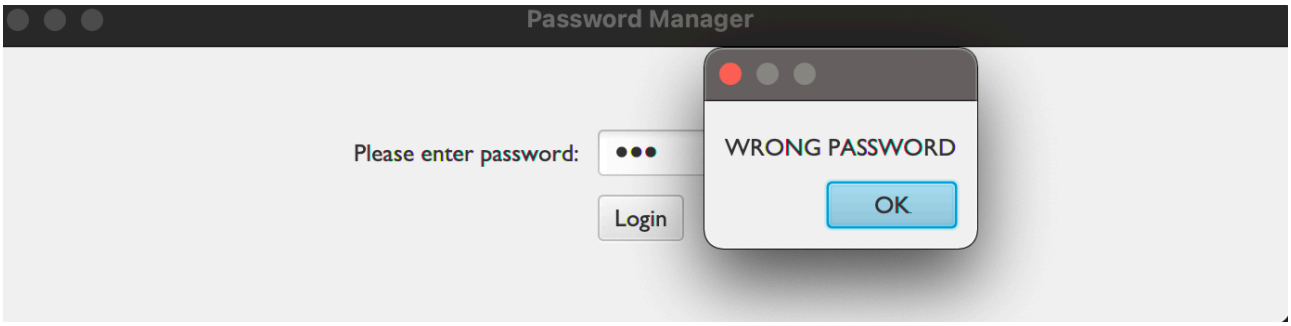

Figure 3: Trying to enter with a wrong password

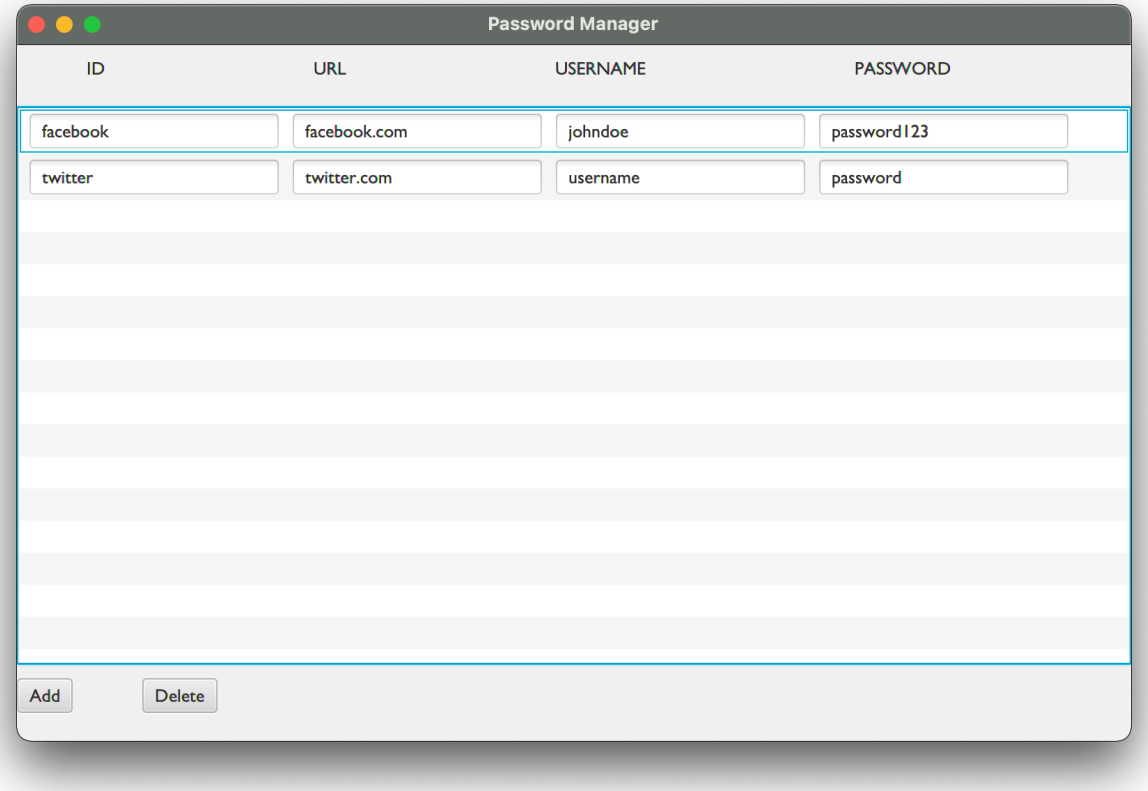

Figure 4: Inside of the password manager

Appendices

https://drive.google.com/file/d/1NCbOKcdDV2wIJ0X5ihd6MVneU9MeWa3n/view?usp=sharing

```
import javafx.application.Application;
import javafx.event.EventHandler;
import javafx.geometry.Insets;
import javafx.geometry.Pos;
import javafx.scene.Scene;
import javafx.scene.control.*;
import javafx.scene.layout.GridPane;
import javafx.stage.Stage;
import javafx.stage.WindowEvent;
import org.bouncycastle.jce.provider.BouncyCastleProvider;

/* main class includes the encryption/decryption of the password information
* it also controls the login page and overall scene transition from login to
password manager
 */
public class Main extends Application {

    private PasswordField password;
    private BouncyCastleProvider provider;
    private PasswordManager manager;

    //this method is called when the javafx application is initialized
    public void start(Stage primaryStage) {
        Label loginPrompt;
        Button login;
        Scene loginScene;
        GridPane grid;

        //add bouncy castle as provider
        provider = new BouncyCastleProvider();

        //create login page
        password = new PasswordField();
        loginPrompt = new Label();
        login = new Button("Login");

        /*this variable checks if the login is done for the first time
        * this is done by checking if an encrypted password information file
exists
        * if the file exists, it is not the first time that the user is logging
in
        * therefore the password is used to decrypt the encrypted password
information
        * if it is indeed the first time the user is logging in,
        * the entered password is used to encrypt a new password information
file
         */
        boolean firstLogin = false;
        if(Security.encryptedFileExists()) {
            loginPrompt.setText("Please enter password:");
        }
        //logging in for the first time
        else {
            loginPrompt.setText("Welcome! Please set a password:");
            firstLogin = true;
        }

        boolean finalFirstLogin = firstLogin;

        //login button
```

```
        login.setOnAction(actionEvent -> {
            //the user can't enter empty password
            if(password.getText().equals("")) {
                Dialog emptyPassword = new Dialog();
                Label label = new Label("YOU DIDN'T ENTER PASSWORD");

emptyPassword.getDialogPane().getButtonTypes().add(ButtonType.OK);
                emptyPassword.getDialogPane().setContent(label);
                emptyPassword.show();
            }
            else {
                //if it is first time logging in OR if the password is correct,
the password manager opens
                if(!finalFirstLogin && Security.decrypt(provider,
password.getText()) || finalFirstLogin) {
                    manager = new PasswordManager();
                    primaryStage.setScene(manager.getScene());
                }
                //wrong password, shows a warning to the user that the password
is incorrect
                else {
                    Dialog wrongPassword = new Dialog();
                    Label label = new Label("WRONG PASSWORD");

wrongPassword.getDialogPane().getButtonTypes().add(ButtonType.OK);
                    wrongPassword.getDialogPane().setContent(label);
                    wrongPassword.show();
                }
            }
        });

        grid = new GridPane();
        grid.setAlignment(Pos.CENTER);
        grid.add(loginPrompt, 0, 0);
        grid.add(password, 1, 0);
        grid.add(login, 1, 1);
        grid.setHgap(10);
        grid.setVgap(10);
        grid.setPadding(new Insets(10, 10, 10, 10));

        loginScene = new Scene(grid, 700, 150);
        primaryStage.setTitle("Password Manager");
        primaryStage.setScene(loginScene);

        //when closing the program, encrypts the new updated information and
deletes the decrypted version
        primaryStage.setOnCloseRequest(new EventHandler<WindowEvent>()
        {
            @Override
            public void handle(WindowEvent windowEvent) {
                if(manager == null) {
                    Security.deleteDecryptedFile();
                }
                else {
                    Security.deleteDecryptedFile();
                    Security.encrypt(provider, password.getText(),
manager.getJsonString());
                }
            }
        });
```

```
        primaryStage.show();
    }
}

import javafx.geometry.Insets;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.control.Label;
import javafx.scene.control.ListView;
import javafx.scene.layout.HBox;
import javafx.scene.layout.VBox;
import org.json.simple.*;
import org.json.simple.parser.*;
import java.nio.file.Files;
import java.nio.file.Paths;
import java.util.ArrayList;

//PasswordManager class includes the actual password manager scene if the login
is successful
public class PasswordManager {

    private Scene scene;
    private ListView list;

    public PasswordManager() {
        HBox labels;
        VBox vbox;
        HBox buttons;
        Button delete, add;

        list = new ListView();
        list.setPrefWidth(1000);
        list.setPrefHeight(400);

        labels = new HBox();
        labels.getChildren().addAll(new Label("ID"), new Label("URL"), new
Label("USERNAME"), new Label("PASSWORD"));
        labels.setSpacing(150);
        labels.setPadding(new Insets(10, 100, 10, 50));

        vbox = new VBox();
        vbox.setSpacing(10);

        buttons = new HBox();
        buttons.setSpacing(50);

        //delete button deletes the selected row
        delete = new Button("Delete");
        delete.setOnAction(actionEvent -> {
            Row selectedRow = (Row) list.getSelectionModel().getSelectedItem();
            list.getItems().remove(selectedRow);
            delete.setVisible(false);
        });

        //delete will only be visible when a row is selected
        delete.setVisible(false);
        list.setOnMouseClicked(mouseEvent -> {
            delete.setVisible(true);
        });
```

```
        //add button adds a new row
        add = new  Button("Add");
        add.setOnAction(actionEvent -> {
            list.getItems().add(new Row());
        });

        //gets the information from the decrypted json file
        getInformation();

        buttons.getChildren().addAll(add, delete);
        vbox.getChildren().addAll(labels, list, buttons);
        scene = new Scene(vbox, 800, 500);
    }

    public Scene getScene() {
        return scene;
    }

    //gets the information from the decrypted json file where all the password
information are stored
    //adds the information to the list to be shown on the screen
    private boolean getInformation() {

        JSONParser parser = new JSONParser();
        try {
            //read the file
            byte[] bytesRead = Files.readAllBytes(Paths.get("files/
information.json"));
            Object jsonArray = parser.parse(new String(bytesRead, "UTF-8"));

            //add the parsed information to the list
            JSONArray storedPasswords = (JSONArray)jsonArray;

            for(int i = 0; i < storedPasswords.size(); i++) {
                JSONObject jsonObject = (JSONObject)storedPasswords.get(i);
                String id = (String)jsonObject.get("id");
                String url = (String)jsonObject.get("url");
                String username = (String)jsonObject.get("username");
                String password = (String)jsonObject.get("password");
                Information info = new Information(id, url, username, password);
                list.getItems().add(new Row(info));
            }
        } catch(Exception e) {
            e.printStackTrace();
            return false;
        }
        return true;
    }

    //this method gets the edited information from the list
    // and returns the json object as a string
    // so that it can be encrypted in Main class
    public String getJsonString() {
        String jsonString;
        ArrayList<Information> infoArray = new ArrayList<Information>();

        //traverse the list to get the all updated information
        for (Object item: list.getItems()) {
            Row row = (Row) item;
            infoArray.add(row.getInformation());
```

```
        }

        //create the json array to have the string in same format as json
        JSONArray jsonInfoArray = new JSONArray();
        for (int i = 0;i < infoArray.size() ; i++) {
            JSONObject information = new JSONObject();
            information.put("id", infoArray.get(i).getId());
            information.put("url",  infoArray.get(i).getUrl());
            information.put("username",  infoArray.get(i).getUsername());
            information.put("password",  infoArray.get(i).getPassword());
            jsonInfoArray.add(information);
        }

        jsonString = jsonInfoArray.toString();
        return jsonString;
    }
}
```

```
import javafx.scene.control.TextField;
import javafx.scene.layout.HBox;

//Row class is an HBox where the information for the specific url/username/
password is displayed

public class Row extends HBox {

    private TextField id, url, username, password;

    //constructors
    public Row() {
        id = new TextField();
        url = new TextField();
        username = new TextField();
        password = new TextField();
        this.setSpacing(10);
        this.getChildren().addAll(id, url, username, password);
    }

    public Row(Information information) {
        id = new TextField(information.getId());
        url = new TextField(information.getUrl());
        username = new TextField(information.getUsername());
        password = new TextField(information.getPassword());
        this.setSpacing(10);
        this.getChildren().addAll(id, url, username, password);
    }

    public Information getInformation() {
        return new Information(id.getText(), url.getText(), username.getText(),
password.getText());
    }
}
```

```
import org.bouncycastle.jce.provider.BouncyCastleProvider;
import org.bouncycastle.util.encoders.Hex;

import javax.crypto.Cipher;
import javax.crypto.SecretKey;
```

```
import javax.crypto.SecretKeyFactory;
import javax.crypto.spec.IvParameterSpec;
import javax.crypto.spec.PBEKeySpec;
import java.io.File;
import java.nio.file.Files;
import java.nio.file.Paths;
import java.security.SecureRandom;

public class Security {

    //encrypts the password information file
    public static boolean encrypt(BouncyCastleProvider provider, String
password, String jsonString) {
        try {
            //generate random iv
            SecureRandom secureRandom = SecureRandom.getInstance("DEFAULT",
provider);
            byte[] generatedIV = new byte[16];
            secureRandom.nextBytes(generatedIV);

            //generate key based on the user password entered during the login
            byte[] salt = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14,
15};
            PBEKeySpec keySpec = new PBEKeySpec(password.toCharArray(), salt,
5000, 128);
            SecretKeyFactory factory =
SecretKeyFactory.getInstance("PBKDF2WITHHMACSHA256", provider);
            SecretKey key = factory.generateSecret(keySpec);

            Cipher cipher = Cipher.getInstance("AES/CBC/PKCS5Padding",
provider);
            cipher.init(Cipher.ENCRYPT_MODE, key, new
IvParameterSpec(generatedIV));

            //in order to encrypt, the data string first needs to be written
into pdf
            Files.write(Paths.get("files/passwords.pdf"),
jsonString.getBytes());
            byte[] input = Files.readAllBytes(Paths.get("files/passwords.pdf"));
            byte[] output = cipher.doFinal(input);

            //write the encrypted version
            String outFile = "files/information." + Hex.toHexString(generatedIV)
+ ".aes";
            Files.write(Paths.get(outFile), output);

            //delete the pdf including the passwords
            File pdfFile = new File("files/passwords.pdf");
            pdfFile.delete();
        }
        catch (Exception e) {
            return false;
        }
        return true;
    }

    //decrypts the password information file using the entered password
    //if the file cannot be decrypted, false is returned
    public static boolean decrypt(BouncyCastleProvider provider, String
password) {
```

```
        File[] files = new File("files").listFiles();
        File encryptedFile = new File("");
        String[] splittedName = new String[5]; //the size is determined as 5
even though the file is expected to contain
        // 3 words when it is split from the "." to give margin to error
        //find the encrypted file
        for (File file : files) {
            if (file.isFile()) {
                String fname = file.getName();
                splittedName = fname.split("\\.");
                if (splittedName[splittedName.length - 1].equals("aes")) {
                    encryptedFile = file;
                    System.out.println(encryptedFile.getName());
                    break;
                }
            }
        }
        if (!encryptedFile.exists()) {
            System.out.println("File not found!");
            return false;
        } else {
            try {
                Cipher cipher = Cipher.getInstance("AES/CBC/PKCS5Padding",
provider);

                //read the file
                byte[] input = Files.readAllBytes(Paths.get("files/" +
encryptedFile.getName()));

                //get the iv from the file name
                String ivString = splittedName[1];
                byte[] iv = Hex.decode(ivString);

                //get key from password
                byte[] salt = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14,
15};
                PBEKeySpec keySpec = new PBEKeySpec(password.toCharArray(),
salt, 5000, 128);
                SecretKeyFactory factory =
SecretKeyFactory.getInstance("PBKDF2WITHHMACSHA256", provider);
                SecretKey key = factory.generateSecret(keySpec);

                cipher.init(Cipher.DECRYPT_MODE, key, new IvParameterSpec(iv));
                byte[] output = cipher.doFinal(input);

                //write the decrypted file as a json file
                String outFile = "files/" + splittedName[0] + ".json";
                Files.write(Paths.get(outFile), output);

                //delete the encrypted file (because the new information will be
encrypted again with a different iv)
                encryptedFile.delete();

            } catch (Exception e) {
                return false;
            }
        }
        return true;
    }
```

```
    //deletes the decrypted password information
    public static void deleteDecryptedFile() {
        File[] files = new File("files").listFiles();
        for (File file : files) {
            if (file.isFile()) {
                String[] names = file.getName().split("\\.");
                if (names[names.length - 1].equals("json")) {
                    file.delete();
                }
            }
        }
    }

    //checks if there is an encrypted file
    public static boolean encryptedFileExists() {
        File[] files = new File("files").listFiles();
        for (File file : files) {
            if (file.isFile()) {
                System.out.println(file.getName());
                System.out.println(file.getName().length());
                int charCount = 0;
                for(int i = 0; i < file.getName().length(); i++) {
                    if(file.getName().charAt(i) == '.') {
                        charCount++;
                    }
                }
                System.out.println(charCount);
                String[] names = (file.getName()).split("\\.");
                System.out.println(names.length);
                if(names.length > 0) {
                    if (names[names.length - 1].equals("aes"))
                        return true;
                }
            }
        }
        return false;
    }
}

/* Information class
* Contains the information shown on each row of the password manager
* This information includes:
* id: An id to identify the specific password/username/website combination
* url
* username
* password
 */
public class Information {
    //these variables are private to ensure encapsulation
    private String id, url, username, password;

    //constructors
    public Information(String id, String url, String username, String password)
{
        this.id = id;
        this.url = url;
        this.username = username;
        this.password = password;
    }
```

```
    // getter/setter functions for the private variables
    public String getId() {
        return id;
    }

    public String getUrl() {
        return url;
    }

    public String getUsername() {
        return username;
    }

    public String getPassword() {
        return password;
    }
}
```

TEST CODE:

```
import org.bouncycastle.jce.provider.BouncyCastleProvider;
import org.junit.jupiter.api.AfterEach;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;

import java.io.File;

import static org.junit.jupiter.api.Assertions.assertEquals;

public class SecurityTest {

    @BeforeEach
    void encrypt() {
        Security.encrypt(new BouncyCastleProvider(), "password", "{[]}");
    }

    @Test
    void wrongDecrypt() {
        assertEquals(true,Security.decrypt(new BouncyCastleProvider(),
"password"));

    }

    @Test
    void rightDecrypt() {
        assertEquals(false,Security.decrypt(new BouncyCastleProvider(),
"wrongpassword"));
    }

    @AfterEach
    void deleteFiles() {
        File[] files = new File("files").listFiles();
        for (File file : files) {
            if (file.isFile()) {
                file.delete();
            }
        }
    }
}
```