

|  |
| --- |
| МИНОБРНАУКИ РОССИИ |
| Федеральное государственное бюджетное образовательное учреждение высшего образования  **«МИРЭА – Российский технологический университет» РТУ МИРЭА** |

Институт КБСП направление

Кафедра КБ-4 «Интеллектуальные системы информационной безопасности» Дисциплина « »

**ЗАДАНИЕ НА КУРСОВУЮ РАБОТУ**

Студент: курса, группа:

1. Тема: **Реализация алгоритма шифрования IDEA в desktop-приложении.**
2. Срок представления проекта (работы) к защите г.
3. Исходные данные для разработки

4 Содержание пояснительной записки: Задание

Содержание

Описание предметной области

Список использованных источников Приложения

Руководитель работы Ф.И.О.

подпись, дата

Задание принял к исполнению Ф.И.О.

подпись, дата

Москва г.

**Оглавление**

[1.Основная часть. 4](#_Toc132178077)

[**1.1.Описание работы алгоритма IDEA. 4**](#_Toc132178078)

[**1.2.Описание интерфейса приложения. 5**](#_Toc132178079)

[**1.3.Программная реализация шифровальщика. 6**](#_Toc132178080)

[**1.4.Тестовая часть. 8**](#_Toc132178081)

[2.Заключение. 11](#_Toc132178082)

[3.Список литературы. 12](#_Toc132178083)

[4.Приложение А. 13](#_Toc132178084)

**Введение.**

В текущей информационной эпохе безопасность конфиденциальной информации играет критически важную роль. Для обеспечения надежной защиты используются различные методы шифрования. Один из самых распространенных методов шифрования - создание шифровальщика файлов на основе симметричного блочного алгоритма IDEA, который использует блоки размером 128 бит и ключи переменной длины. В данной курсовой работе исследуется создание desktop-приложения под операционную систему Windows, которое обеспечивает защиту конфиденциальной информации с помощью алгоритма IDEA. Основная задача заключается в разработке программного обеспечения, которое позволит пользователям защитить их файлы и папки от несанкционированного доступа. Работа будет охватывать все основные этапы разработки, включая исследование особенностей алгоритма, создание пользовательского интерфейса и тестирование. Этот шифровальщик файлов может быть применен в повседневной жизни, для защиты личной и организационной информации.

# Основная часть.

IDEA (International Data Encryption Algorithm) – блочный шифр симметричного типа, который был разработан компанией ETH Zurich в 1991 году [1]. Он широко применяется в различных системах защиты данных и в системах электронной коммерции.

Главное преимущество IDEA заключается в его безопасности, так как данный алгоритм достаточно сложен для взлома методами перебора [2]. Он использует мощное и тщательно проверенное сочетание смены медленного шифрования и быстрой арифметики.

## Описание работы алгоритма IDEA.

Метод шифрования IDEA является блочным шифром симметричного типа, то есть использует один и тот же ключ для шифрования и дешифрования данных. Он работает с блоками данных по 64 бита (8 байт). Для процесса шифрования используется 128-битный ключ, который обрабатывается путем многократного раундирования данных [3].

Алгоритм IDEA состоит из нескольких основных этапов. Первый этап - это генерация раундовых ключей. Генерация ключей происходит из основного пароля, который вводится пользователем и который сам по себе не используется для шифрования данных. Начальный пароль обрабатывается различными методами, включая генерацию подключа и сдвигов, для получения раундовых ключей, которые будут использоваться в процессе шифрования.

Следующим этапом является сам процесс шифрования. IDEA использует последовательность раундов, которые многократно применяются к данным для того, чтобы сделать их более невосприимчивыми к взлому. В каждом раунде происходит операция подстановки, перемешивания и складывания. В конечном итоге, после выполнения всех раундов, данные будут зашифрованы и будут выглядеть случайным образом.

IDEA поддерживает несколько режимов шифрования данных. В режиме ECB (Electronic Codebook) каждый блок данных обрабатывается независимо, что делает данную схему очень быстрой, однако, это также делает ее больше всего подверженной взлому из всех режимов. В режиме CBC (Cipher Block Chaining) каждый блок данных зависит от предыдущего, что делает этот режим более безопасным, однако, менее производительным. Режим CFB (Cipher Feedback) позволяет шифровать данные произвольной длины, при этом, существует задержка в передаче данных. Режим OFB (Output Feedback) - позволяет шифровать данные произвольной длины, и не имеет задержки в передаче данных, однако, это делает его менее безопасным, чем режим CBC.

## Описание интерфейса приложения.

При разработке desktop-приложения, пользовательский интерфейс имеет ключевое значение. Программа должна быть удобной и простой в использовании для обычных пользователей. Интерфейс должен предоставлять пользователю выбор файлов (или папок), которые будут зашифрованы, а также настройки шифрования. Важно убедиться, что пользователи могут легко понимать, как использовать приложение, и что оно дает им достаточно гибкости для настройки параметров шифрования.

В дальнейшем наше desktop-приложение будет содержать в себе следующие элементы интерфейса:

* Выбор файлов для шифрования. Пользователи смогут выбирать какие файлы нужно зашифровать и в какую папку сохранить результат работы шифровальщика.
* Кнопки для выбора режима работы IDEA. Пользователи могут выбирать режимы CBC, ECB, CFB и OFB.
* Кнопки для выбора шифрования и дешифрования.
* Ввод ключа. Пользователи должны ввести ключ шифрования, который используется для защиты файлов. Мы также можем предоставить пользователю возможность генерации ключа.
* Вспомогательная информация. Мы можем добавить сообщения об ошибках, индикатор прогресса и другую вспомогательную информацию в интерфейс программы.

Приложение IDEA должно быть легко использовать и иметь простой интерфейс без дополнительных хлопот для пользователя. Его задача - обеспечить защиту пользовательских файлов и папок, а не увеличивать сложность использования компьютера.

Для создания интерфейса был выбран JavaFX – платформа на основе Java для создания насыщенного графического интерфейса в приложениях. На Рис. 1 представлен готовый интерфейс шифровальщика.

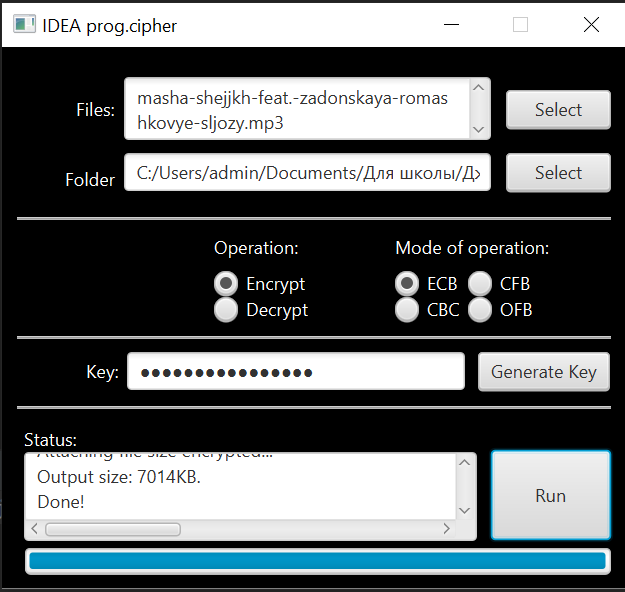


Рис. 1 – интерфейс приложения

## Программная реализация шифровальщика.

\*Все дальнейшие упоминания классов и кода см. Приложении А.

Внешний вид приложения был написан в layout.fxml, но для взаимодействия с пользователем нужно написать еще и контроллер, реализован в Controller.java, который отвечает за прослушивание действий: нажатие кнопок, переключение режимов шифрования, - и обратную связь с пользователем через шкалу прогресса и поле статуса выполнения программы.

Сам же алгоритм шифрования IDEA реализован в отдельном пакете cipher и разбит на подпакеты и отдельные классы. В пакете cipher.modes реализованы четыре режима шифрование OFB, CBC, CFB, ECB в одноименных классах. В пакете cipher.crypto написано три класса: BlockCipher.java – является абстрактным классом для наследования любого класса алгоритма блочного шифрования; Idea.java – класс, в котором реализованы базовые (основные) функции для работы алгоритма блочного шифрования IDEA; MathAlgo.java – в этом классе реализованы функции побайтового изменения информации; OperationMode.java – класс для выбора одного из четырех режима шифрования; ThreadPool.java – класс пула потоков, который отвечает за обработку каждой задачи отдельным потоком; Cipher.java – главный класс приложения, который отвечает за обработку отдельного файла: чтение файла, вызов алгоритма шифрования для блока прочитанной информации, запись измененного блока информации в новый файл, сохранение нового файла, закрытие исходного и нового файлов, а так же отсылает всю обратную связь пользователю относительно изменения файла.

В классе Controller.java, после нажатия на кнопку Run, вызывается функция handleRun() (см. Приложение А, package prog.view, Controller.java). В ней по очереди для каждого файла формируется задача task класса Cipher со следующими параметрами – исходный файл для шифрования, результирующий файл (файл с измененной информацией), ключ, действия шифрование/дешифрование, режим работы алгоритма – CBC, OFB, ECB, FCB. Такая задача task помещается в очередь пула потоков, где ее обрабатывает освободившийся поток. За счет работы пула потоков создается многопоточность, которая ускоряет работу шифровальщика.

## Тестовая часть.

Во время выполнения приложения были сделаны flame graph (см. Рис. 2) и замеры потребления памяти на 1, 5 и 10 минутах работы приложения (см. Рис. 3.1, Рис. 3.2, Рис. 3.3 соответственно), а также зафиксировано количество объектов в хипе (см. Рис. 4).

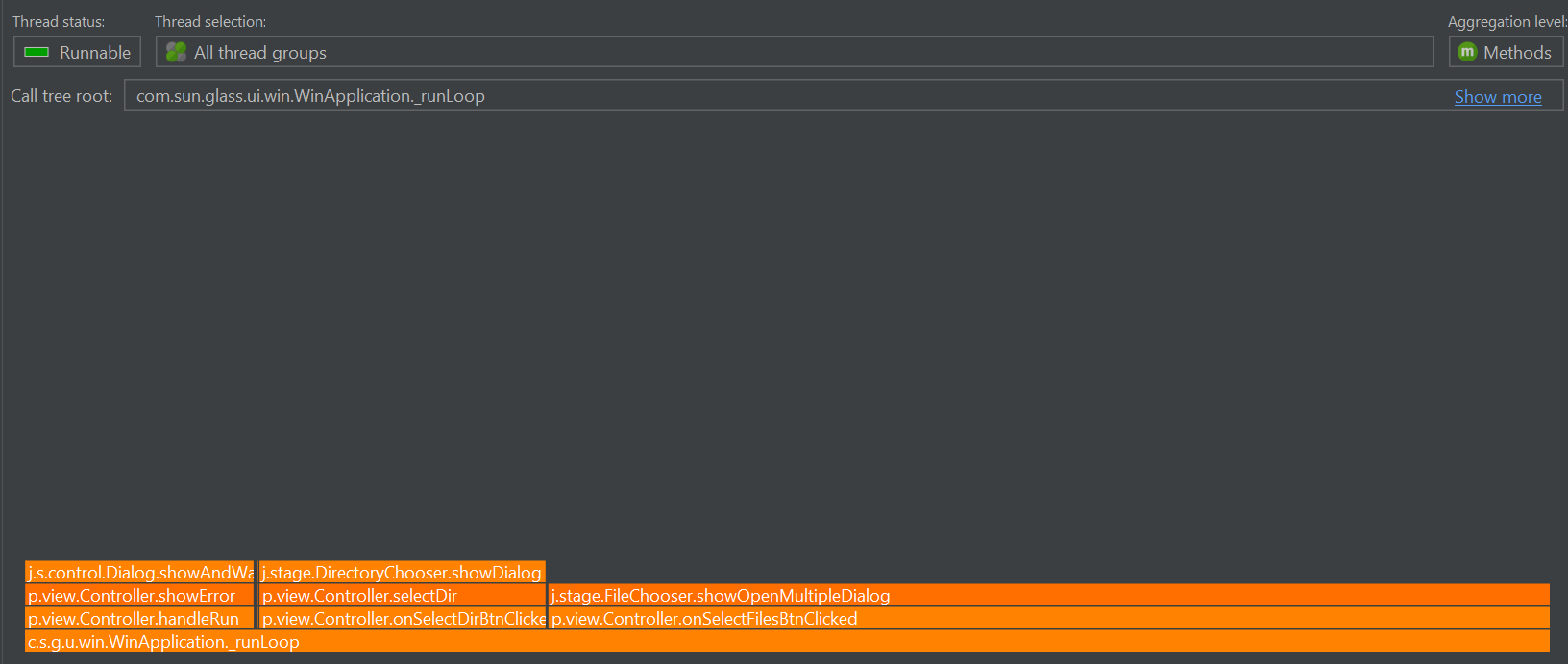


Рис. 2 – flame graph.

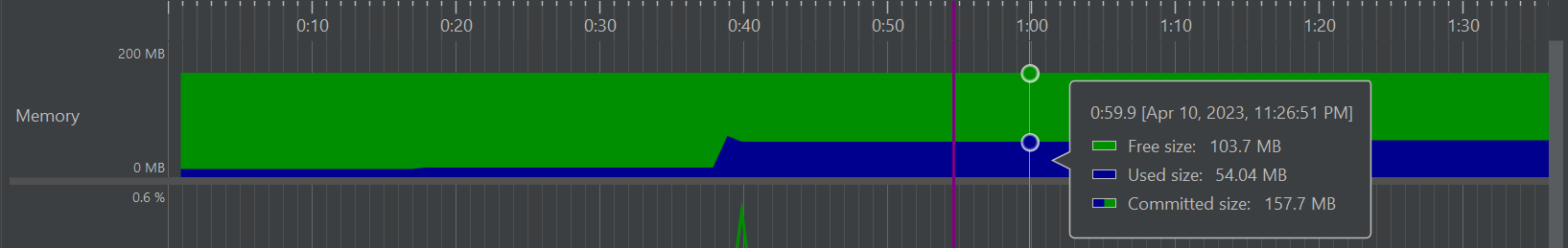


Рис. 3.1 – хип памяти на 1-ой минуте.

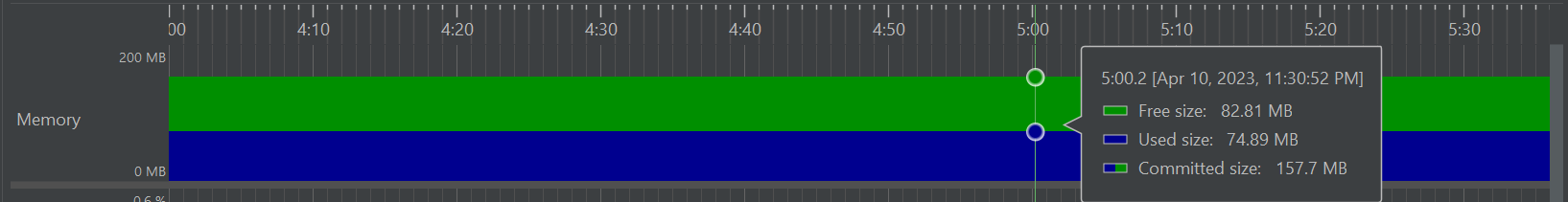


Рис. 3.2 – хип памяти на 5-ой минуте.



Рис. 3.3 – хип памяти на 10-ой минуте.

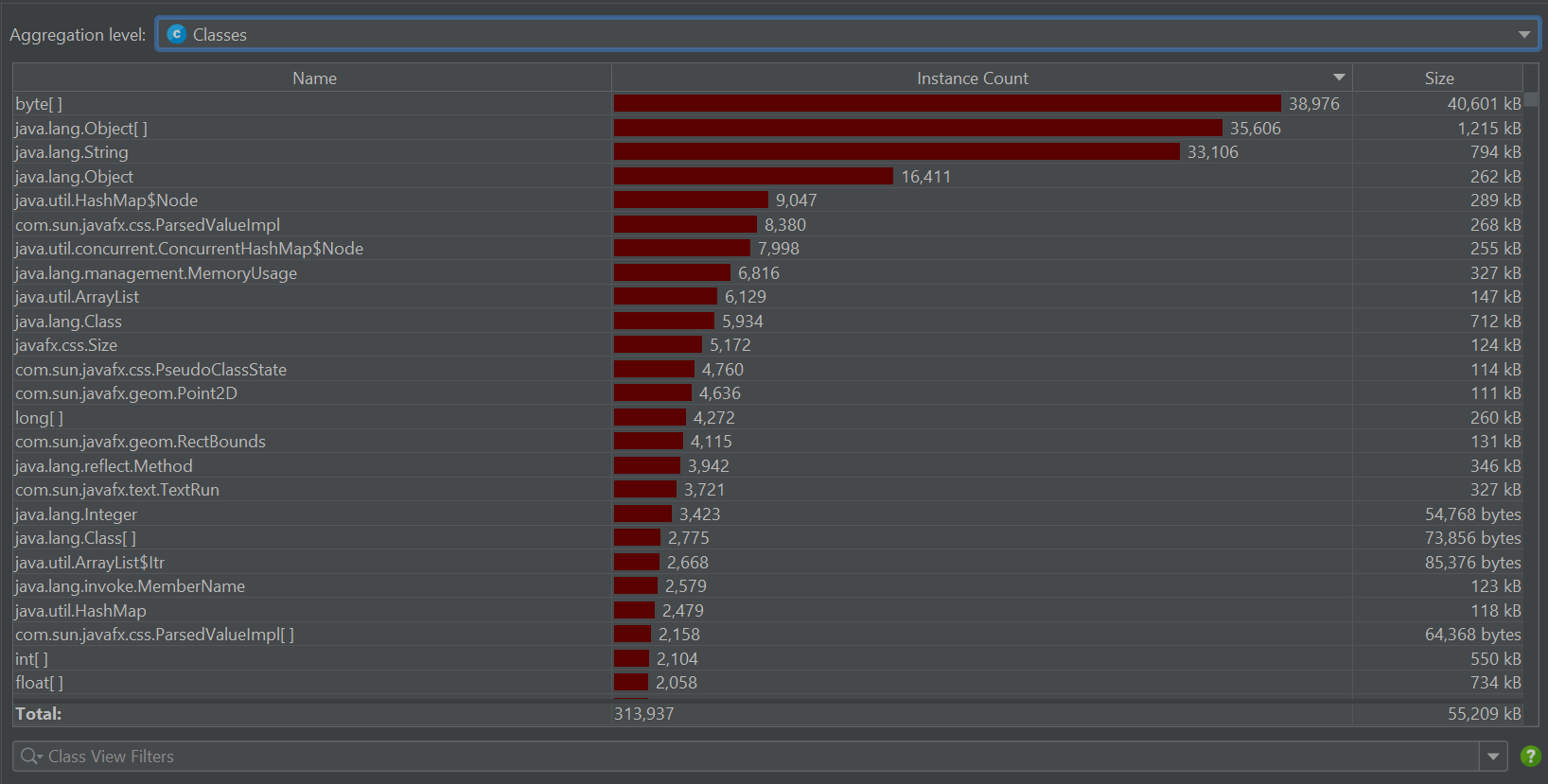


Рис. 4 – размер/количество объектов в хипе.

Так же были сделаны замеры потоков на всем протяжение работы приложения (см Рис. 5), где хорошо видно, что во время выполнения шифрования были созданы дополнительны четыре потока.

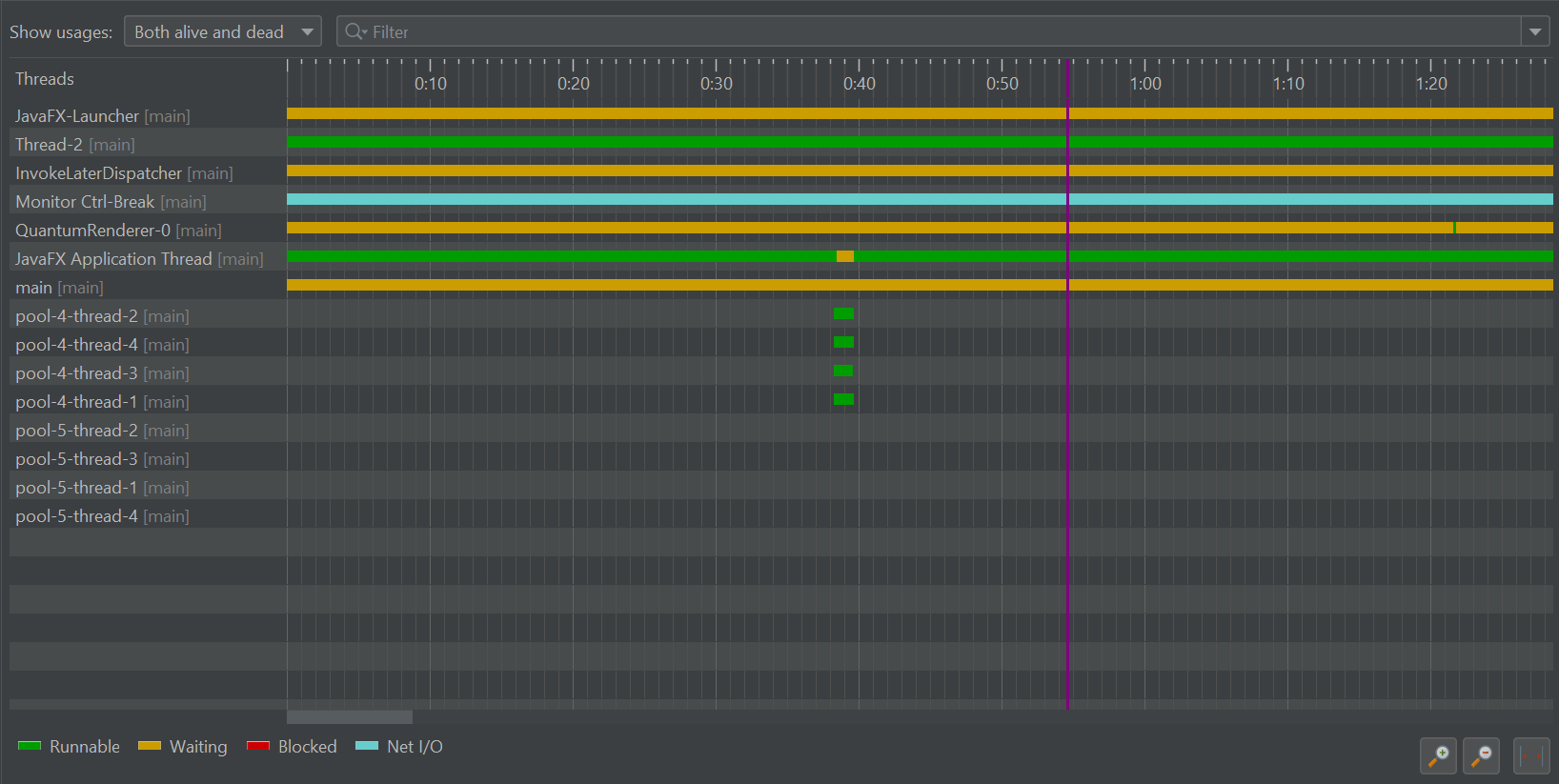


Рис. 5 – работа потоков.

Код программы прошел через статистический анализ исходного кода (см. Рис. 6), где были выявлены незначительные ошибки.

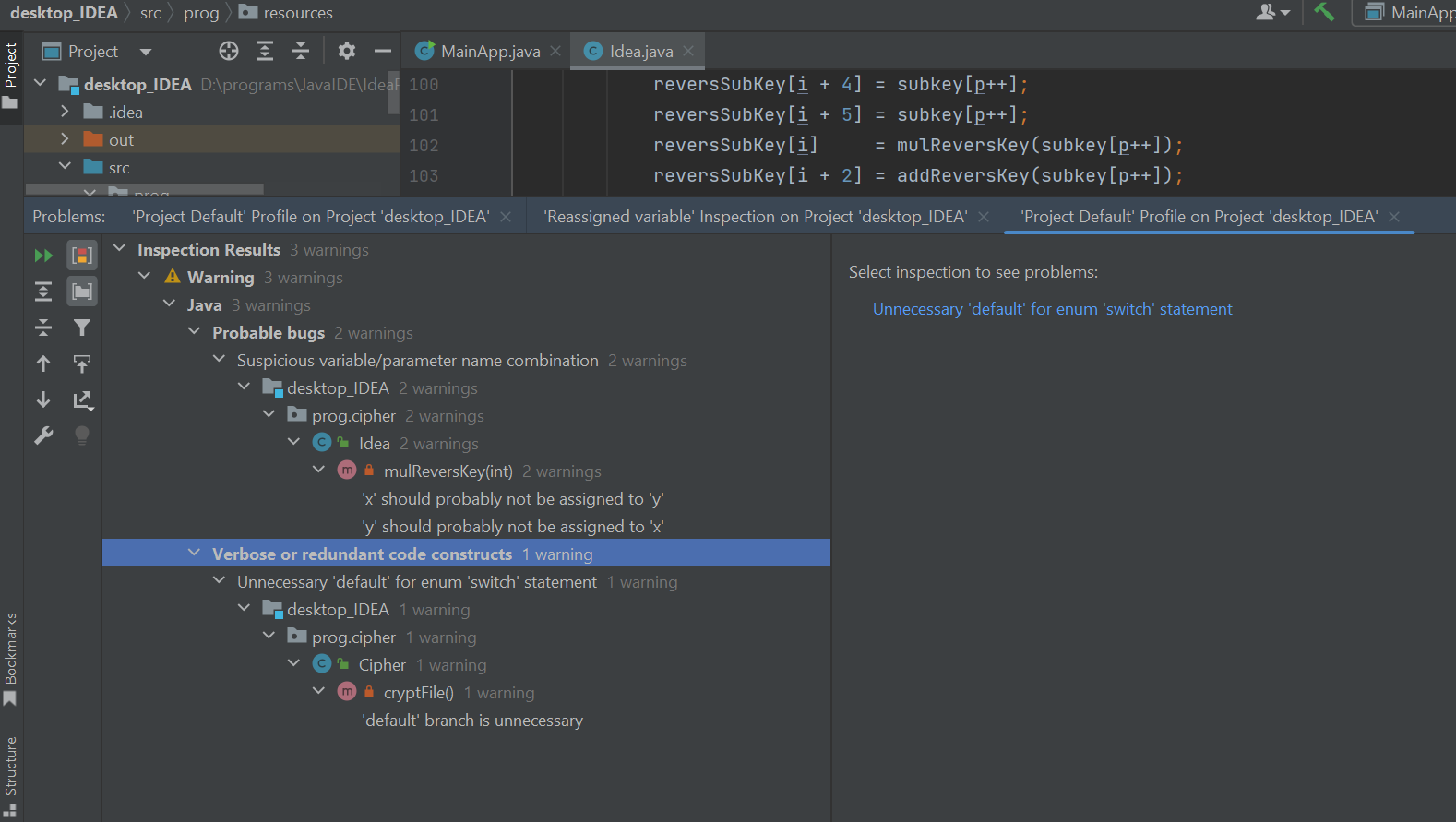


Рис. 6 - статистический анализ исходного кода.

# Заключение.

В заключение, можно сказать, что desktop-приложение шифровальщика файлов на основе алгоритма IDEA - это современный способ защиты конфиденциальной информации от несанкционированного доступа. Создание такого приложения для операционной системы Windows может помочь пользователям быстро и просто зашифровать свои файлы и папки, предоставляя при этом высокий уровень защиты. В ходе разработки приложения были рассмотрены основные этапы, такие как исследование алгоритма IDEA, создание пользовательского интерфейса и тестирование программного обеспечения. Такое приложение может быть использовано как в личных, так и в организационных целях, так как позволяет хранить и защищать конфиденциальный контент, составляющий важную часть личной или организационной информации.

# Список литературы.

1. [Шнайер Б.](https://ru.wikipedia.org/wiki/%D0%A8%D0%BD%D0%B0%D0%B9%D0%B5%D1%80,_%D0%91%D1%80%D1%8E%D1%81" \o "Шнайер, Брюс) Прикладная криптография. Протоколы, алгоритмы, исходные тексты на языке Си = Applied Cryptography. Protocols, Algorithms and Source Code in C. — М.: Триумф, 2002. — 816 с.
2. W. Meier, HTL. Brugg-Windisch, Switzerland. On the Security of the IDEA Block Cipher // Семинар по теории и применению криптографических техник в работе комиссии Advances in Сryptology EUROCRYPT '93 = Workshop on the theory and application of cryptographic techniques on Advances in Сryptology EUROCRYPT '93 Proceedings. — Secaucus, NJ, USA: Springer-Verlag New York, Inc, 1994. — P. 371—385.
3. Xuejia Lai and James Massey. Предложение нового блочного стандарта шифрования = A Proposal for a New Block Encryption Standard, [EUROCRYPT](https://ru.wikipedia.org/w/index.php?title=EUROCRYPT&action=edit&redlink=1) 1990. — Springer-Verlag, 1991. — P. 389—404.

# Приложение А.

**Package prog.**

**MainApp.java**

package prog;  
  
import javafx.application.Application;  
import javafx.fxml.FXMLLoader;  
import javafx.scene.Parent;  
import javafx.scene.Scene;  
import javafx.stage.Stage;  
  
public class MainApp extends Application {  
 @SuppressWarnings("DataFlowIssue")  
 @Override  
 public void start(Stage primaryStage) throws Exception {  
 final Parent root = FXMLLoader.*load*(getClass().getResource("resources/layout.fxml"));  
 final Scene scene = new Scene(root);  
 primaryStage.setTitle("IDEA prog.cipher");  
 primaryStage.setResizable(false);  
 primaryStage.setScene(scene);  
 primaryStage.sizeToScene();  
 primaryStage.centerOnScreen();  
 primaryStage.show();  
 }  
  
 public static void main(String[] args) {  
 *launch*(args);  
 }  
}

**Package prog.view**

**Controller.java**

package prog.view;  
  
import javafx.application.Platform;  
import javafx.stage.DirectoryChooser;  
import prog.cipher.Cipher;  
import prog.cipher.OperationMode;  
import javafx.concurrent.Worker;  
import javafx.fxml.FXML;  
import javafx.scene.control.\*;  
import javafx.stage.FileChooser;  
import javafx.stage.Stage;  
import prog.cipher.ThreadPool;  
  
import java.io.File;  
import java.security.SecureRandom;  
import java.util.ArrayList;  
import java.util.List;  
import java.util.concurrent.ExecutionException;  
  
public class Controller {  
  
 @FXML  
 private TextArea selectedFilesTextArea;  
 @FXML  
 private Button selectInput, selectOutput;  
 @FXML  
 private ToggleGroup operation, operationMode;  
 @FXML  
 private TextField outputFile;  
 @FXML  
 private PasswordField key;  
 @FXML  
 private Button run, GenerateKey;  
 @FXML  
 private RadioButton decrypt, encrypt;  
 @FXML  
 private RadioButton ecb, cbc, ofb, cfb;  
 @FXML  
 private ProgressBar progressBar;  
 @FXML  
 private TextArea status;  
 private final List<File> selectedFiles = new ArrayList<>();  
 private Cipher task;  
 private File output;  
  
 @FXML  
 private void initialize() {  
 outputFile.setText(System.*getProperty*("user.home").replace("\\", "/"));  
 status.appendText("Select files, choose parameters and press run...");  
 }  
  
 */\*\*  
 \* Select input file.  
 \*/* @FXML  
 public void onSelectFilesBtnClicked() {  
 FileChooser fc = new FileChooser();  
 fc.setTitle("Select files");  
 List<File> list = (fc.showOpenMultipleDialog(null));  
 selectedFiles.clear();  
 selectedFiles.addAll(list);  
 updateSelectedFilesTextArea();  
 }  
  
 */\*\*  
 \* Select output file.  
 \*/* @FXML  
 private void onSelectDirBtnClicked() {  
 File f = output != null ? selectDir(output.getParent()) :  
 selectDir();  
 if (f != null) {  
 output = f;  
 outputFile.setText(output.toString().replace("\\", "/"));  
 }  
 }  
  
 */\*\*  
 \* Open a FileChooser to select a file.  
 \*  
 \* @param path path to open  
 \* @return selected file  
 \*/* private File selectDir(String path) {  
 Stage primaryStage = (Stage) outputFile.getScene().getWindow();  
 DirectoryChooser chooser = new DirectoryChooser();  
 chooser.setInitialDirectory(new File(path));  
 chooser.setTitle("Select output");  
 return chooser.showDialog(primaryStage);  
 }  
  
 */\*\*  
 \* Open a FileChooser to select a file in the default path (user.home).  
 \*/* private File selectDir() {  
 return selectDir(System.*getProperty*("user.home"));  
 }  
 private void updateSelectedFilesTextArea() {  
 StringBuilder sb = new StringBuilder();  
 for (File file : selectedFiles) {  
 sb.append(file.getName())  
 .append(System.*lineSeparator*());  
 }  
 selectedFilesTextArea.setText(sb.toString());  
 }  
  
 */\*\*  
 \* Run prog.cipher.  
 \*/* @FXML  
 private void handleRun() {  
 if(handleCancelTask()){  
 blockUI(false);  
 return;  
 }  
 if (selectedFiles.isEmpty()) {  
 showError("no-file");  
 return;  
 } else if (key.getText().equals("")) {  
 showError("no-key");  
 return;  
 }  
 blockUI(true);  
 boolean encrypt = (((RadioButton) operation.getSelectedToggle()).getText()).equals("Encrypt");  
 OperationMode.Mode mode = switch (((RadioButton) operationMode.getSelectedToggle()).getText()) {  
 case "ECB" -> OperationMode.Mode.*ECB*;  
 case "CBC" -> OperationMode.Mode.*CBC*;  
 case "CFB" -> OperationMode.Mode.*CFB*;  
 case "OFB" -> OperationMode.Mode.*OFB*;  
 default -> null;  
 };  
 resetStatus();  
 ThreadPool pool = new ThreadPool(4);  
 for(File in: selectedFiles) {  
 File newFile = new File(output, in.getName());  
 task = new Cipher(in.getPath(), newFile.getPath(), key.getText(), encrypt, mode);  
 task.getStatus().addListener((observable, oldValue, newValue) -> Platform.*runLater*(() -> println(newValue)));  
 task.setOnSucceeded(event -> blockUI(false));  
 progressBar.progressProperty().bind(task.progressProperty());  
 task.setOnFailed(event -> {  
 if(task.getException() != null) {  
 println("Error: " + task.getException().getMessage());  
 }  
 blockUI(false);  
 });  
 pool.addTask(task);  
 }  
 pool.waitAllTasks();  
 pool.stop();  
 }  
  
 */\*\*  
 \* Clear the status box.  
 \*/* private void resetStatus() {  
 status.clear();  
 status.appendText("Let's go!");  
 }  
  
 */\*\*  
 \* Disable or enable the interface controls.  
 \*  
 \* @param running true: disable / false: enable  
 \*/* private void blockUI(boolean running) {  
 if(running) {  
 run.setText("Cancel");  
 } else {  
 run.setText("Run");  
 }  
 selectInput.setDisable(running);  
 selectOutput.setDisable(running);  
 ToggleGroup[] groups = {operation, operationMode};  
 for(ToggleGroup g : groups){  
 for (Toggle t : g.getToggles()) {  
 if(t instanceof RadioButton){  
 ((RadioButton) t).setDisable(running);  
 } else {  
 ((RadioMenuItem) t).setDisable(running);  
 }  
 }  
 }  
 key.setDisable(running);  
 }  
  
 */\*\*  
 \* Generate Key.  
 \*  
 \*/* @FXML  
 public void onGenerateKeyBtnClicked() {  
 String generatedKey = *generateKey*();  
 key.setText(generatedKey);  
 }  
  
 private static String generateKey() {  
 final String CHARACTERS = "0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz";  
 final int KEY\_LENGTH = 16;  
 SecureRandom random = new SecureRandom();  
 StringBuilder key = new StringBuilder(KEY\_LENGTH);  
 for (int i = 0; i < KEY\_LENGTH; i++) {  
 int index = random.nextInt(CHARACTERS.length());  
 char randomChar = CHARACTERS.charAt(index);  
 key.append(randomChar);  
 }  
 return key.toString();  
 }  
  
 */\*\*  
 \* Cancel task.  
 \*  
 \* @return true if the cancel was successful  
 \*/* private boolean handleCancelTask() {  
 boolean canceled = false;  
 if(task != null && task.getState() == Worker.State.*RUNNING*) {  
 println("The operation was cancelled!");  
 canceled = task.cancel();  
 }  
 return canceled;  
 }  
  
 private void println(String msg) {  
 status.appendText("\n" + msg);  
 }  
  
 */\*\*  
 \* Open an alert box to show the error.  
 \*/* private void showError(String error) {  
 Alert alert = new Alert(Alert.AlertType.*ERROR*);  
 alert.setTitle("Error");  
 if (error.equals("no-file")) {  
 alert.setHeaderText("No file chosen");  
 alert.setContentText("You have to choose the file to encrypt.");  
 } else if (error.equals("no-key")) {  
 alert.setHeaderText("No key");  
 alert.setContentText("You have to enter a key.");  
 }  
 alert.showAndWait();  
 }  
  
}

**Package prog.resources**

**Layout.fxml**

*<?*xml version="1.0" encoding="UTF-8"*?>  
  
<?*import javafx.geometry.\**?>  
<?*import javafx.scene.control.\**?>  
<?*import javafx.scene.layout.\**?>*<BorderPane prefHeight="361.0" prefWidth="416.0" style="-fx-background-color: black;" xmlns="http://javafx.com/javafx/17.0.2-ea" xmlns:fx="http://javafx.com/fxml/1" fx:controller="prog.view.Controller">  
 <center>  
 <VBox prefHeight="361.0" prefWidth="408.0" BorderPane.alignment="CENTER">  
 <padding>  
 <Insets bottom="10.0" left="10.0" right="10.0" top="10.0" />  
 </padding>  
 <GridPane prefHeight="95.0" prefWidth="404.0">  
 <columnConstraints>  
 <ColumnConstraints hgrow="SOMETIMES" maxWidth="283.0" minWidth="10.0" prefWidth="62.0" />  
 <ColumnConstraints hgrow="SOMETIMES" maxWidth="526.0" minWidth="10.0" prefWidth="252.0" />  
 <ColumnConstraints hgrow="SOMETIMES" maxWidth="526.0" minWidth="10.0" prefWidth="72.0" />  
 </columnConstraints>  
 <rowConstraints>  
 <RowConstraints minHeight="10.0" prefHeight="30.0" vgrow="SOMETIMES" />  
 <RowConstraints minHeight="10.0" prefHeight="30.0" vgrow="SOMETIMES" />  
 </rowConstraints>  
 <Label alignment="CENTER\_RIGHT" contentDisplay="RIGHT" prefHeight="17.0" prefWidth="65.0" text="Files:" textFill="WHITE" />  
 <TextArea fx:id="selectedFilesTextArea" prefHeight="32.0" prefWidth="247.0" wrapText="true" GridPane.columnIndex="1">  
 <GridPane.margin>  
 <Insets left="5.0" right="5.0" />  
 </GridPane.margin>  
 </TextArea>  
 <Button fx:id="selectInput" mnemonicParsing="false" onAction="#onSelectFilesBtnClicked" prefHeight="25.0" prefWidth="70.0" text="Select" GridPane.columnIndex="2" GridPane.halignment="RIGHT" />  
 <Label alignment="CENTER\_RIGHT" contentDisplay="RIGHT" layoutX="10.0" layoutY="20.0" prefHeight="41.0" prefWidth="65.0" text="Folder" textFill="WHITE" GridPane.rowIndex="1">  
 <GridPane.margin>  
 <Insets top="10.0" />  
 </GridPane.margin>  
 </Label>  
 <TextField fx:id="outputFile" editable="false" GridPane.columnIndex="1" GridPane.rowIndex="1">  
 <GridPane.margin>  
 <Insets left="5.0" right="5.0" />  
 </GridPane.margin>  
 </TextField>  
 <Button fx:id="selectOutput" mnemonicParsing="false" onAction="#onSelectDirBtnClicked" prefHeight="25.0" prefWidth="70.0" text="Select" GridPane.columnIndex="2" GridPane.halignment="RIGHT" GridPane.rowIndex="1" />  
 <padding>  
 <Insets top="10.0" />  
 </padding>  
 </GridPane>  
 <Separator prefHeight="0.0" prefWidth="402.0">  
 <VBox.margin>  
 <Insets top="10.0" />  
 </VBox.margin>  
 </Separator>  
 <GridPane>  
 <columnConstraints>  
 <ColumnConstraints hgrow="SOMETIMES" minWidth="10.0" prefWidth="100.0" />  
 </columnConstraints>  
 </GridPane>  
 <GridPane prefHeight="68.0" prefWidth="355.0">  
 <columnConstraints>  
 <ColumnConstraints hgrow="ALWAYS" maxWidth="140.33333333333331" minWidth="10.0" prefWidth="109.66666666666667" />  
 <ColumnConstraints hgrow="ALWAYS" maxWidth="140.33333333333331" minWidth="10.0" prefWidth="109.66666666666667" />  
 <ColumnConstraints hgrow="ALWAYS" maxWidth="184.33332570393878" minWidth="10.0" prefWidth="143.66666666666663" />  
 </columnConstraints>  
 <rowConstraints>  
 <RowConstraints minHeight="10.0" prefHeight="30.0" vgrow="SOMETIMES" />  
 <RowConstraints minHeight="10.0" prefHeight="30.0" vgrow="SOMETIMES" />  
 </rowConstraints>  
 <Label text="Operation:" textFill="WHITE" GridPane.columnIndex="1">  
 <GridPane.margin>  
 <Insets />  
 </GridPane.margin>  
 <padding>  
 <Insets left="10.0" />  
 </padding>  
 </Label>  
 <Label text="Mode of operation:" textFill="WHITE" GridPane.columnIndex="2" GridPane.valignment="CENTER">  
 <padding>  
 <Insets left="10.0" />  
 </padding></Label>  
 <VBox GridPane.columnIndex="1" GridPane.rowIndex="1">  
 <GridPane.margin>  
 <Insets left="10.0" />  
 </GridPane.margin>  
 <RadioButton fx:id="encrypt" mnemonicParsing="false" selected="true" text="Encrypt" textFill="WHITE">  
 <toggleGroup>  
 <ToggleGroup fx:id="operation" />  
 </toggleGroup>  
 </RadioButton>  
 <RadioButton fx:id="decrypt" mnemonicParsing="false" text="Decrypt" textFill="WHITE" toggleGroup="$operation" />  
 </VBox>  
 <HBox GridPane.columnIndex="2" GridPane.rowIndex="1">  
 <padding>  
 <Insets left="10.0" />  
 </padding>  
 <VBox>  
 <padding>  
 <Insets right="5.0" />  
 </padding>  
 <RadioButton fx:id="ecb" mnemonicParsing="false" selected="true" text="ECB" textFill="WHITE">  
 <toggleGroup>  
 <ToggleGroup fx:id="operationMode" />  
 </toggleGroup>  
 </RadioButton>  
 <RadioButton fx:id="cbc" mnemonicParsing="false" text="CBC" textFill="WHITE" toggleGroup="$operationMode" />  
 </VBox>  
 <VBox>  
 <RadioButton fx:id="cfb" mnemonicParsing="false" text="CFB" textFill="WHITE" toggleGroup="$operationMode" />  
 <RadioButton fx:id="ofb" mnemonicParsing="false" text="OFB" textFill="WHITE" toggleGroup="$operationMode" />  
 </VBox>  
 </HBox>  
  
 </GridPane>  
 <Separator layoutX="20.0" layoutY="81.0" prefWidth="200.0">  
 <opaqueInsets>  
 <Insets />  
 </opaqueInsets>  
 <padding>  
 <Insets top="10.0" />  
 </padding>  
 </Separator>  
 <GridPane>  
 <columnConstraints>  
 <ColumnConstraints hgrow="SOMETIMES" maxWidth="74.0" minWidth="10.0" prefWidth="70.0"/>  
 <ColumnConstraints hgrow="SOMETIMES" maxWidth="310.0" minWidth="10.0" prefWidth="238.0"/>  
 <ColumnConstraints hgrow="SOMETIMES" maxWidth="295.00000762939453" minWidth="10.0"  
 prefWidth="93.33333333333331"/>  
 </columnConstraints>  
 <rowConstraints>  
 <RowConstraints minHeight="10.0" prefHeight="30.0" vgrow="SOMETIMES"/>  
 </rowConstraints>  
 <PasswordField fx:id="key" prefHeight="25.0" prefWidth="248.0" promptText="Password"  
 GridPane.columnIndex="1" GridPane.rowIndex="0">  
 <GridPane.margin>  
 <Insets left="5.0" right="5.0"/>  
 </GridPane.margin>  
 </PasswordField>  
 <Button fx:id="GenerateKey" mnemonicParsing="false" onAction="#onGenerateKeyBtnClicked"  
 prefHeight="25.0" prefWidth="88.0" text="Generate Key" GridPane.columnIndex="2"  
 GridPane.halignment="RIGHT" GridPane.rowIndex="0"/>  
 <padding>  
 <Insets top="5.0"/>  
 </padding>  
 <Label alignment="CENTER\_RIGHT" contentDisplay="RIGHT" text="Key:" textFill="WHITE"  
 GridPane.halignment="RIGHT"/>  
 </GridPane>  
  
 <Separator layoutX="20.0" layoutY="81.0" prefWidth="200.0">  
 <opaqueInsets>  
 <Insets />  
 </opaqueInsets>  
 <padding>  
 <Insets top="10.0" />  
 </padding>  
 </Separator>  
 <HBox alignment="TOP\_RIGHT">  
 <opaqueInsets>  
 <Insets />  
 </opaqueInsets>  
 <VBox.margin>  
 <Insets top="10.0" />  
 </VBox.margin>  
 <VBox prefHeight="77.0" prefWidth="307.0">  
 <Label text="Status:" textFill="WHITE">  
 <VBox.margin>  
 <Insets left="5.0" />  
 </VBox.margin></Label>  
 <TextArea fx:id="status" editable="false" prefHeight="60.0" prefWidth="290.0">  
 <VBox.margin>  
 <Insets left="5.0" />  
 </VBox.margin></TextArea>  
 </VBox>  
 <Button fx:id="run" minHeight="60.0" minWidth="80.0" mnemonicParsing="false" onAction="#handleRun" prefHeight="60.0" prefWidth="80.0" text="Run">  
 <HBox.margin>  
 <Insets left="10.0" top="16.0" />  
 </HBox.margin>  
 </Button>  
 </HBox>  
 <ProgressBar fx:id="progressBar" prefHeight="25.0" prefWidth="398.0" progress="0.0" VBox.vgrow="ALWAYS">  
 <padding>  
 <Insets top="5.0" />  
 </padding>  
 <VBox.margin>  
 <Insets left="5.0" />  
 </VBox.margin>  
 </ProgressBar>  
 </VBox>  
 </center>  
</BorderPane>

**Package prog.cipher**

**Cipher.java**

package prog.cipher;  
  
import prog.cipher.modes.CBC;  
import prog.cipher.modes.CFB;  
import prog.cipher.modes.ECB;  
import prog.cipher.modes.OFB;  
import javafx.beans.property.SimpleStringProperty;  
import javafx.beans.property.StringProperty;  
import javafx.concurrent.Task;  
  
import java.io.IOException;  
import java.nio.ByteBuffer;  
import java.nio.channels.FileChannel;  
import java.nio.file.Paths;  
import java.nio.file.StandardOpenOption;  
import java.util.Arrays;  
  
public class Cipher extends Task<Void> {  
  
 public final static int *KEY\_SIZE* = 16;  
 public final static int *BLOCK\_SIZE* = 8;  
 public final static int *ROUNDS* = 8;  
 private final String input;  
 private final String output;  
 private final String key;  
 private final boolean encrypt;  
 private final OperationMode.Mode mode;  
 private final StringProperty status; *// To print messages in status box* public Cipher(String input, String output, String key, boolean encrypt, OperationMode.Mode mode) {  
 this.input = input;  
 this.output = output;  
 this.key = key;  
 this.encrypt = encrypt;  
 this.mode = mode;  
 status = new SimpleStringProperty();  
 }  
  
 public StringProperty getStatus() {  
 return status;  
 }  
  
 */\*\*  
 \* Encrypts / decrypts file.  
 \*/* private void cryptFile() throws Exception {  
 try (FileChannel inChannel = FileChannel.*open*(Paths.*get*(input), StandardOpenOption.*READ*);  
 FileChannel outChannel = FileChannel.*open*(Paths.*get*(output), StandardOpenOption.*CREATE*,  
 StandardOpenOption.*TRUNCATE\_EXISTING*, StandardOpenOption.*WRITE*)) {  
  
  
 OperationMode opMod = switch (mode) {  
 case *ECB* -> new ECB(encrypt, key);  
 case *CBC* -> new CBC(encrypt, key);  
 case *CFB* -> new CFB(encrypt, key);  
 case *OFB* -> new OFB(key);  
 default -> throw new IllegalArgumentException("Incorrect mode of operation.");  
 };  
 status.setValue((encrypt ? "Encrypting" : "Decrypting") + " file is: " + input + ". With " + mode + " mode.");  
  
  
 long inFileSize = inChannel.size();  
 long inDataLen, outDataLen;  
 if (encrypt) {  
 inDataLen = inFileSize;  
 outDataLen = (inDataLen + *BLOCK\_SIZE* - 1) / *BLOCK\_SIZE* \* *BLOCK\_SIZE*;  
 status.setValue("Input size: " + inDataLen / 1024 + "KB.");  
 } else {  
 if (inFileSize == 0) {  
 throw new IOException("Input file is empty.");  
 } else if (inFileSize % *BLOCK\_SIZE* != 0) {  
 throw new IOException("Input file size is not a multiple of " + *BLOCK\_SIZE* + ".");  
 }  
 inDataLen = inFileSize - *BLOCK\_SIZE*;  
 outDataLen = inDataLen;  
 status.setValue("Input size: " + (inDataLen + *BLOCK\_SIZE*) / 1024 + "KB.");  
 }  
  
 *// Encrypt / decrypt data* status.setValue("Running IDEA...");  
 long t0 = System.*currentTimeMillis*();  
 processData(inChannel, inDataLen, outChannel, outDataLen, opMod);  
 long tf = (System.*currentTimeMillis*() - t0);  
 status.setValue((encrypt ? "Encryption" : "Decryption") + " finished (" + tf + "ms)");  
  
 if (encrypt) {  
 status.setValue("Attaching file size encrypted...");  
 writeData(outChannel, inDataLen, opMod);  
 status.setValue("Output size: " + inDataLen / 1024 + "KB.");  
 } else {  
 status.setValue("Checking file size...");  
 long dataSize = readData(inChannel, opMod);  
 if (dataSize < 0 || dataSize > inDataLen || dataSize < inDataLen - *BLOCK\_SIZE* + 1) {  
 throw new IOException("Input file is not a valid cryptogram (wrong file size)");  
 }  
 if (dataSize != outDataLen) {  
 outChannel.truncate(dataSize);  
 status.setValue("Truncating output file...");  
 }  
 status.setValue("Output size: " + dataSize / 1024 + "KB.");  
 }  
 status.setValue("Done!");  
 }  
 }  
  
 */\*\*  
 \* Read the input file in chunks of 2MB, encrypt/decrypt the chunks and write it in the output file.  
 \*/* private void processData(FileChannel inChannel, long inDataLen, FileChannel outChannel, long outDataLen,  
 OperationMode opMod) throws IOException {  
 final int bufSize = 0x200000; *// 2MB of buffer* ByteBuffer buf = ByteBuffer.*allocate*(bufSize);  
 long filePos = 0;  
 while (filePos < inDataLen) {  
 updateProgress(filePos, inDataLen);  
 int bytesToRead = (int) Math.*min*(inDataLen - filePos, bufSize);  
 buf.limit(bytesToRead);  
 buf.position(0);  
 int bytesRead = inChannel.read(buf);  
 if (bytesRead != bytesToRead) {  
 throw new IOException("Incomplete data chunk read from file.");  
 }  
  
 int chunkLen = (bytesRead + *BLOCK\_SIZE* - 1) / *BLOCK\_SIZE* \* *BLOCK\_SIZE*;  
 Arrays.*fill*(buf.array(), bytesRead, chunkLen, (byte) 0);  
 for (int pos = 0; pos < chunkLen; pos += *BLOCK\_SIZE*) {  
 opMod.crypt(buf.array(), pos);  
 }  
  
 int bytesToWrite = (int) Math.*min*(outDataLen - filePos, chunkLen);  
 buf.limit(bytesToWrite);  
 buf.position(0);  
 int bytesWritten = outChannel.write(buf);  
 if (bytesWritten != bytesToWrite) {  
 throw new IOException("Incomplete data chunk written to file.");  
 }  
 filePos += chunkLen;  
 }  
 }  
  
 */\*\*  
 \* Write the length of the encrypted data in an encrypted block at the end of the file.  
 \* The length is package is 8-byte block, this block is encrypted and finally added at the end  
 \* of output file.  
 \*/* private void writeData(FileChannel outChannel, long dataLength, OperationMode opMod)  
 throws IOException {  
 byte[] block = packageData(dataLength);  
 opMod.crypt(block);  
 ByteBuffer buf = ByteBuffer.*wrap*(block);  
 int bytesWritten = outChannel.write(buf);  
 if (bytesWritten != *BLOCK\_SIZE*) {  
 throw new IOException("Error while writing data length suffix.");  
 }  
 }  
  
 */\*\*  
 \* Get the length of the data that was encrypted.  
 \* This data is saved encrypted in the last block of the cryptogram.  
 \* Read the last block of the file, decrypt block and unpackage data length.  
 \*/* private long readData(FileChannel channel, OperationMode opMod) throws IOException {  
 ByteBuffer buf = ByteBuffer.*allocate*(*BLOCK\_SIZE*);  
 int bytesRead = channel.read(buf);  
 if (bytesRead != *BLOCK\_SIZE*) {  
 throw new IOException("Unable to read data length suffix.");  
 }  
 byte[] block = buf.array();  
  
 opMod.crypt(block);  
  
 return unpackageData(block);  
 }  
  
 */\*\*  
 \* Packs 45-bit number into an 8-byte block. Used to encode the file size.  
 \*/* private byte[] packageData(long size) {  
 if (size > 0x1FFFFFFFFFFFL) { *// 45 bits -> 32TB* throw new IllegalArgumentException("File too long.");  
 }  
 byte[] b = new byte[*BLOCK\_SIZE*];  
 b[7] = (byte) (size << 3);  
 b[6] = (byte) (size >> 5);  
 b[5] = (byte) (size >> 13);  
 b[4] = (byte) (size >> 21);  
 b[3] = (byte) (size >> 29);  
 b[2] = (byte) (size >> 37);  
 return b;  
 }  
  
 */\*\*  
 \* Extracts a 45-bit number from an 8-byte block. Used to decode the file size.  
 \* Returns -1 if the encoded value is invalid. This means that the input file is not a valid cryptogram.  
 \*/* private long unpackageData(byte[] b) {  
 if (b[0] != 0 || b[1] != 0 || (b[7] & 7) != 0) {  
 return -1;  
 }  
 return (long) (b[7] & 0xFF) >> 3 |  
 (long) (b[6] & 0xFF) << 5 |  
 (long) (b[5] & 0xFF) << 13 |  
 (long) (b[4] & 0xFF) << 21 |  
 (long) (b[3] & 0xFF) << 29 |  
 (long) (b[2] & 0xFF) << 37;  
 }  
  
 @Override  
 protected Void call() throws Exception {  
 updateProgress(0, 1);  
 cryptFile();  
 updateProgress(1, 1);  
 return null;  
 }  
}

**OperationMode.java**

package prog.cipher;  
  
import prog.cipher.crypto.Idea;  
  
public abstract class OperationMode {  
  
 public enum Mode {  
 *ECB*, *CBC*, *CFB*, *OFB* }  
  
 protected final Idea idea;  
  
 protected final boolean encrypt;  
  
 public OperationMode(Idea idea, boolean encrypt) {  
 this.idea = idea;  
 this.encrypt = encrypt;  
 }  
  
 protected abstract void crypt(byte[] data, int pos);  
  
 void crypt(byte[] data){  
 crypt(data, 0);  
 }  
  
 public boolean isEncrypt() {  
 return encrypt;  
 }  
}

**ThreadPool.java**

package prog.cipher;  
  
import java.util.concurrent.\*;  
  
public class ThreadPool {  
  
 private final ExecutorService executorService;  
  
 public ThreadPool(int threadsNumber) {  
 executorService = Executors.*newFixedThreadPool*(threadsNumber);  
 }  
  
 public void addTask(Cipher task) {  
 executorService.submit(task);  
 }  
  
 public void waitAllTasks() {  
 try {  
 executorService.shutdown();  
 executorService.awaitTermination(Long.*MAX\_VALUE*, TimeUnit.*NANOSECONDS*);  
 } catch (InterruptedException e) {  
 Thread.*currentThread*().interrupt();  
 }  
 }  
  
 public void stop() {  
 executorService.shutdownNow();  
 }  
}

**Package prog.cipher.crypto**

**BlockCipher.java**

package prog.cipher.crypto;  
  
public abstract class BlockCipher {  
  
 private final int keySize;  
 private final int blockSize;  
  
 BlockCipher(int keySize, int blockSize) {  
 this.keySize = keySize;  
 this.blockSize = blockSize;  
 }  
  
 public int getBlockSize() {  
 return blockSize;  
 }  
  
 protected abstract void setKey(byte[] key);  
  
 protected void setKey(String charKey) {  
 setKey(MathAlgo.*makeKey*(charKey, keySize));  
 }  
  
 public abstract void crypt(byte[] data, int offset);  
  
 public void crypt(byte[] data) {  
 crypt(data, 0);  
 }  
}

**Idea.java**

package prog.cipher.crypto;  
  
import static prog.cipher.Cipher.\*;  
import static prog.cipher.crypto.MathAlgo.*glue2Bytes*;  
  
public class Idea extends BlockCipher {  
  
 private final boolean encrypt;  
 private int[] subKey;  
  
 public Idea(String charKey, boolean encrypt) {  
 super(*KEY\_SIZE*, *BLOCK\_SIZE*);  
 this.encrypt = encrypt;  
 setKey(charKey);  
 }  
  
 protected void setKey(byte[] key) {  
 int[] tempSubKey = generateSubKeys(key);  
 if (encrypt) {  
 subKey = tempSubKey;  
 } else {  
 subKey = invertSubKeys(tempSubKey);  
 }  
 }  
  
 public void crypt(byte[] data, int i) {  
  
 int d1 = *glue2Bytes*(data[i], data[i + 1]);  
 int d2 = *glue2Bytes*(data[i + 2], data[i + 3]);  
 int d3 = *glue2Bytes*(data[i + 4], data[i + 5]);  
 int d4 = *glue2Bytes*(data[i + 6], data[i + 7]);  
  
 int j = 0;  
 for (int round = 0; round < *ROUNDS*; round++) {  
 int a = mul(d1, subKey[j++]);  
 int b = add(d2, subKey[j++]);  
 int c = add(d3, subKey[j++]);  
 int d = mul(d4, subKey[j++]);  
 int e = a ^ c;  
 int f = b ^ d;  
  
 int g = mul(e, subKey[j++]);  
 int s = add(f, g);  
 int h = mul(s, subKey[j++]);  
 int k = add(g, h);  
  
 d1 = a ^ h;  
 d2 = c ^ h;  
 d3 = b ^ k;  
 d4 = d ^ k;  
 }  
  
 int r0 = mul(d1, subKey[j++]);  
 int r1 = add(d3, subKey[j++]);  
 int r2 = add(d2, subKey[j++]);  
 int r3 = mul(d4, subKey[j]);  
  
 data[i] = (byte) (r0 >> 8);  
 data[i + 1] = (byte) r0;  
 data[i + 2] = (byte) (r1 >> 8);  
 data[i + 3] = (byte) r1;  
 data[i + 4] = (byte) (r2 >> 8);  
 data[i + 5] = (byte) r2;  
 data[i + 6] = (byte) (r3 >> 8);  
 data[i + 7] = (byte) r3;  
 }  
  
 private int[] generateSubKeys(byte[] userKey) {  
 if (userKey.length != 16) {  
 throw new IllegalArgumentException();  
 }  
 int[] key = new int[*ROUNDS* \* 6 + 4];  
  
 int b1, b2;  
 for (int i = 0; i < userKey.length / 2; i++) {  
 key[i] = *glue2Bytes*(userKey[2 \* i], userKey[2 \* i + 1]);  
 }  
  
 for (int i = userKey.length / 2; i < key.length; i++) {  
 b1 = key[(i + 1) % 8 != 0 ? i - 7 : i - 15] << 9;  
 b2 = key[(i + 2) % 8 < 2 ? i - 14 : i - 6] >>> 7;  
 key[i] = (b1 | b2) & 0xFFFF;  
 }  
 return key;  
 }  
  
 private int[] invertSubKeys(int[] subkey) {  
 int[] reversSubKey = new int[subkey.length];  
 int p = 0;  
 int i = *ROUNDS* \* 6;  
  
 reversSubKey[i] = mulReversKey(subkey[p++]);  
 reversSubKey[i + 1] = addReversKey(subkey[p++]);  
 reversSubKey[i + 2] = addReversKey(subkey[p++]);  
 reversSubKey[i + 3] = mulReversKey(subkey[p++]);  
  
 for (int r = *ROUNDS* - 1; r > 0; r--) {  
 i = r \* 6;  
 reversSubKey[i + 4] = subkey[p++];  
 reversSubKey[i + 5] = subkey[p++];  
 reversSubKey[i] = mulReversKey(subkey[p++]);  
 reversSubKey[i + 2] = addReversKey(subkey[p++]);  
 reversSubKey[i + 1] = addReversKey(subkey[p++]);  
 reversSubKey[i + 3] = mulReversKey(subkey[p++]);  
 }  
  
 reversSubKey[4] = subkey[p++];  
 reversSubKey[5] = subkey[p++];  
 reversSubKey[0] = mulReversKey(subkey[p++]);  
 reversSubKey[1] = addReversKey(subkey[p++]);  
 reversSubKey[2] = addReversKey(subkey[p++]);  
 reversSubKey[3] = mulReversKey(subkey[p]);  
 return reversSubKey;  
 }  
  
 private int add(int x, int y) {  
 return (x + y) & 0xFFFF;  
 }  
  
 private int addReversKey(int x) {  
 return (0x10000 - x) & 0xFFFF;  
 }  
  
 private int mul(int x, int y) {  
 long m = (long) x \* y;  
 if (m != 0) {  
 return (int) (m % 0x10001) & 0xFFFF;  
 } else {  
 if (x != 0 || y != 0) {  
 return (1 - x - y) & 0xFFFF;  
 }  
 return 1;  
 }  
 }  
  
 private int mulReversKey(int x) {  
 if (x <= 1) {  
 return x;  
 }  
 try {  
 int y = 0x10001;  
 int t0 = 1;  
 int t1 = 0;  
 while (true) {  
 t1 += y / x \* t0;  
 y %= x;  
 if (y == 1) {  
 return (1 - t1) & 0xffff;  
 }  
 t0 += x / y \* t1;  
 x %= y;  
 if (x == 1) {  
 return t0;  
 }  
 }  
 } catch (ArithmeticException e) {  
 return 0;  
 }  
 }  
}

**MathAlgo.java**

package prog.cipher.crypto;  
  
public class MathAlgo {  
 public static byte[] makeKey(String charKey, int size) {  
 byte[] key = new byte[size];  
 int i, j;  
 for (j = 0; j < key.length; ++j) {  
 key[j] = 0;  
 }  
 for (i = 0, j = 0; i < charKey.length(); i++, j = (j + 1) % key.length) {  
 key[j] ^= (byte) charKey.charAt(i);  
 }  
 return key;  
 }  
  
 public static void xor(byte[] a, int pos, byte[] b, int blockSize) {  
 for (int p = 0; p < blockSize; p++) {  
 a[pos + p] ^= b[p];  
 }  
 }  
  
 public static int glue2Bytes(int byte1, int byte2) {  
 byte1 = (byte1 & 0xFF) << 8; *// xxxxxxxx00000000* byte2 = byte2 & 0xFF; *// 00000000xxxxxxxx* return (byte1 | byte2); *// xxxxxxxxxxxxxxxx* }  
  
 public static byte[] glue2Bytes(byte[] byte1, byte[] byte2) {  
 byte[] out = new byte[byte1.length + byte2.length];  
 int i = 0;  
 for (byte aB1 : byte1) {  
 out[i++] = aB1;  
 }  
 for (byte aB2 : byte2) {  
 out[i++] = aB2;  
 }  
 return out;  
 }  
}

**Package prog.cipher.modes**

**CBC.java**

package prog.cipher.modes;  
  
import prog.cipher.crypto.Idea;  
import prog.cipher.OperationMode;  
  
import static prog.cipher.crypto.MathAlgo.*makeKey*;  
import static prog.cipher.crypto.MathAlgo.*xor*;  
  
  
public class CBC extends OperationMode {  
  
 private final int blockSize;  
 private byte[] prev;  
 private final byte[] newPrev;  
  
 public CBC(boolean encrypt, String key) {  
 super(new Idea(key, encrypt), encrypt);  
 blockSize = idea.getBlockSize();  
 prev = *makeKey*(key, blockSize);  
 newPrev = new byte[blockSize];  
 }  
  
 @Override  
 protected void crypt(byte[] data, int pos) {  
 if (encrypt) {  
 *xor*(data, pos, prev, blockSize);  
 idea.crypt(data, pos);  
 System.*arraycopy*(data, pos, prev, 0, blockSize);  
 } else {  
 System.*arraycopy*(data, pos, newPrev, 0, blockSize);  
 idea.crypt(data, pos);  
 *xor*(data, pos, prev, blockSize);  
 prev = newPrev.clone();  
 }  
 }  
}

**CFB.java**

package prog.cipher.modes;  
  
import prog.cipher.crypto.Idea;  
import prog.cipher.crypto.MathAlgo;  
import prog.cipher.OperationMode;  
  
import java.util.Arrays;  
  
  
public class CFB extends OperationMode {  
 private final int R = 8;  
 private final int blockSize;  
 private int partSize;  
 private int rounds;  
 private byte[] feedback;  
  
 public CFB(boolean encrypt, String key) {  
 super(new Idea(key, true), encrypt);  
 blockSize = idea.getBlockSize();  
 assert blockSize % R == 0 : "R must be divisor of blockSize";  
 partSize = R;  
 rounds = blockSize / R;  
 feedback = MathAlgo.*makeKey*(key, blockSize);  
 }  
  
 @Override  
 protected void crypt(byte[] data, int pos) {  
 byte[][] block = new byte[rounds][];  
 for (int i = 0; i < rounds; i++) {  
 block[i] = Arrays.*copyOfRange*(data, pos + partSize \* i, pos + partSize \* i + partSize);  
 }  
 byte[][] crypt = new byte[0][];  
 if (!this.isEncrypt()) {  
 crypt = new byte[rounds][];  
 for (int i = 0; i < rounds; i++) {  
 crypt[i] = block[i].clone();  
 }  
 }  
 *// Run CFB algorithm* byte[] feedbackP1, feedbackP2;  
 for (int i = 0; i < rounds; i++) {  
 idea.crypt(feedback);  
 feedbackP1 = Arrays.*copyOfRange*(feedback, 0, partSize);  
 feedbackP2 = Arrays.*copyOfRange*(feedback, partSize, blockSize);  
 MathAlgo.*xor*(block[i], 0, feedbackP1, partSize);  
 if (this.isEncrypt()) {  
 feedback = MathAlgo.*glue2Bytes*(feedbackP2, block[i]);  
 } else {  
 feedback = MathAlgo.*glue2Bytes*(feedbackP2, crypt[i]);  
 }  
 }  
 *// Merge results* for (int i = 0; i < rounds; i++) {  
 System.*arraycopy*(block[i], 0, data, pos + partSize \* i, partSize);  
 }  
 }  
}

**ECB.java**

package prog.cipher.modes;  
  
import prog.cipher.crypto.Idea;  
import prog.cipher.OperationMode;  
  
public class ECB extends OperationMode {  
  
 public ECB(boolean encrypt, String key) {  
 super(new Idea(key, encrypt), encrypt);  
 }  
  
 @Override  
 protected void crypt(byte[] data, int pos) {  
 idea.crypt(data, pos);  
 }  
}

**OFB.java**

package prog.cipher.modes;  
  
import prog.cipher.crypto.Idea;  
import prog.cipher.OperationMode;  
  
import static prog.cipher.crypto.MathAlgo.*makeKey*;  
import static prog.cipher.crypto.MathAlgo.*xor*;  
  
public class OFB extends OperationMode {  
  
 private final int blockSize;  
 private final byte[] feedback;  
  
 public OFB(String key) {  
 super(new Idea(key, true), true);  
 blockSize = idea.getBlockSize();  
 feedback = *makeKey*(key, blockSize);  
 }  
  
 @Override  
 protected void crypt(byte[] data, int pos) {  
 idea.crypt(feedback);  
 *xor*(data, pos, feedback, blockSize);  
 }  
}