magmacrypt

1. Теоретическая часть

2. Практическая часть

2.1. Общие сведения

Данная реализация на языке Java использует версию языка 11 и JDK версии 11. Для сборки приложения используется фреймворк Maven версии 3.6.3; для написания unit-тестов - JUnit версии 4.11.

Сборка и выполнение по умолчанию осуществляются на виртуальной машине с запущенным дистрибутивом Fedora Linux Server 35 (версия ядра - 5.15.10-200.fc35.x86_64).

Вывод команды `java --version` для среды сборки и выполнения по умолчанию:

```
openjdk 11.0.13 2021-10-19
OpenJDK Runtime Environment 18.9 (build 11.0.13+8)
OpenJDK 64-Bit Server VM 18.9 (build 11.0.13+8, mixed mode, sharing)
```

В качестве основных параметров сборки определены следующие свойства:

1. `groupId = ru.mirea.edu.magmacrypt` - для управления компиляцией исходников, также является основным пакетом приложения и используется в качестве соответствующего идентификатора;
2. `artefactId = magmacrypt` и `version = 1.0` - для генерации jar-архива приложения;
3. `mainClass = ru.mirea.edu.magmacrypt.App` - для определения расположения метода `main` (точки входа в приложение);

Структура файлов исходного кода выглядит следующим образом (вывод команды `tree`):

1. Каталог `src` - содержит файлы исходного кода с расширением `.java`:
    1. Каталог `main` - приложение, далее пакет `ru.mirea.edu.magmacrypt` (`gropuId`):
        1. Класс `App` - основной класс приложения;
        2. Пакет `auxiliary` - вспомогательные классы для работы с данными;
        3. Пакет `cipher` - классы, относящиеся к реализации алгоритма шифрования.
    2. Каталог `test` - unit-тесты, далее пакет `ru.mirea.edu.magmacrypt` (`gropuId`), содержащий класс `AppTest` для проверки работы алгоритма шифрования и взаимодействия с различными типами входных данных;
2. Каталог `target` (создается и наполняется при компиляции исходного кода - файлы `.class`, и сборке `jar`-файла):
    1. Каталог `classes`, далее пакет `ru.mirea.edu.magmacrypt` (`gropuId`)
    2. Каталог `test-classes`, далее пакет `ru.mirea.edu.magmacrypt` (`gropuId`)
    3. Файл `magmacrypt-1.0.jar` - собранное приложение;

3. Файл `pom.xml` - конфигурация сборки;
4. Файл `rebuild.sh` - скрипт очистки собранного решения и сборки нового;
5. Каталог `TEST_DATA` - набор данных, используемых для тестирования.

2.2. Описание реализации

Приложение описывают следующие диаграммы классов:

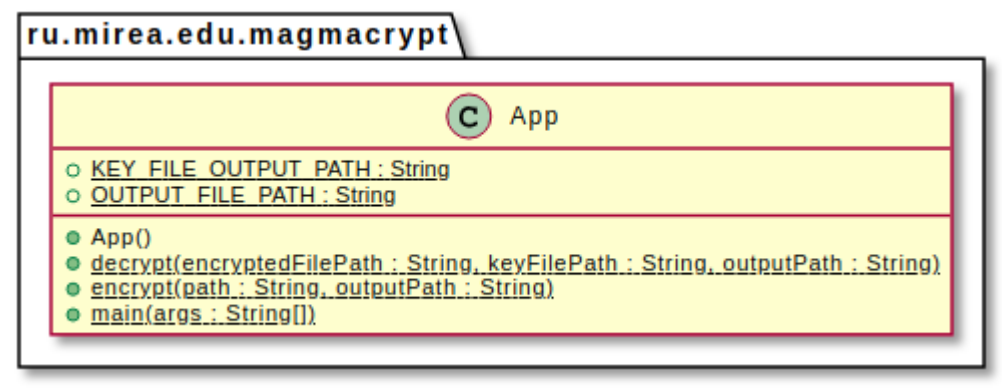

Рис. X: Основной пакет

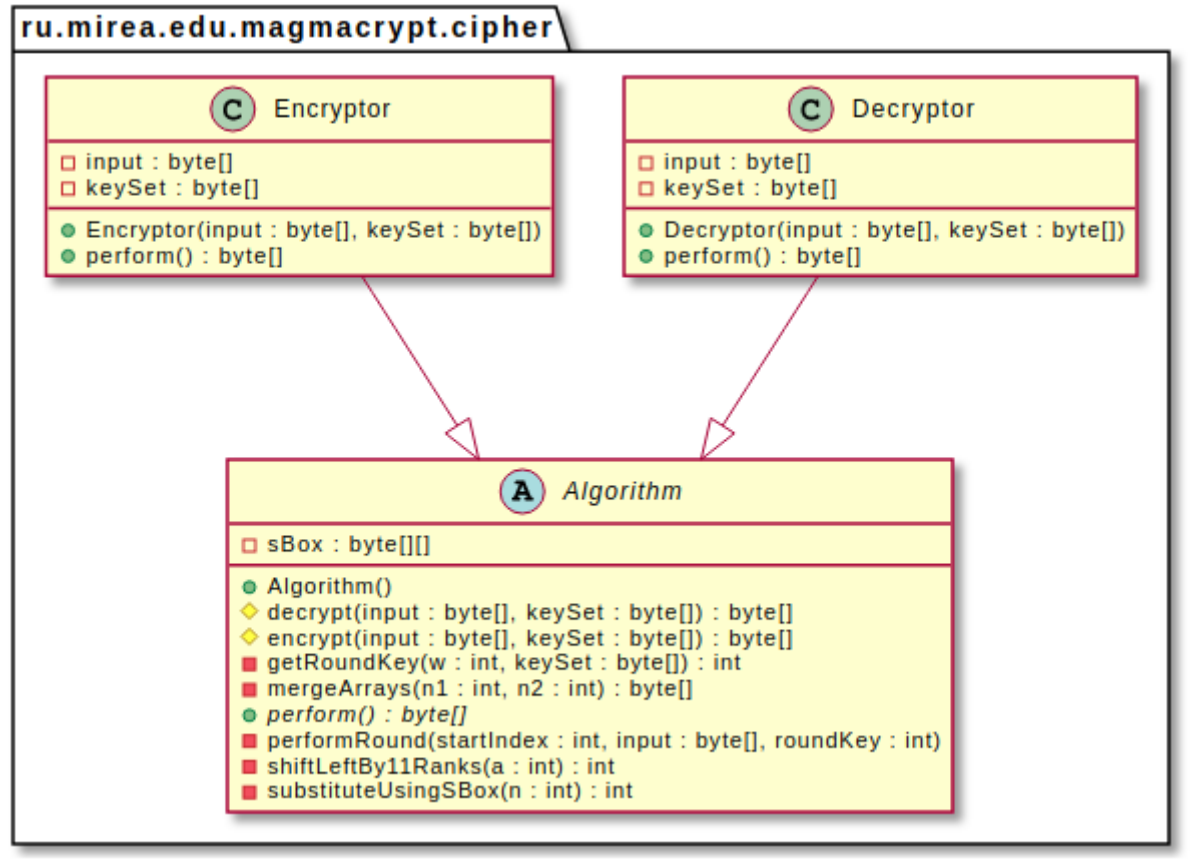

Рис. X: Пакет с реализацией алгоритма

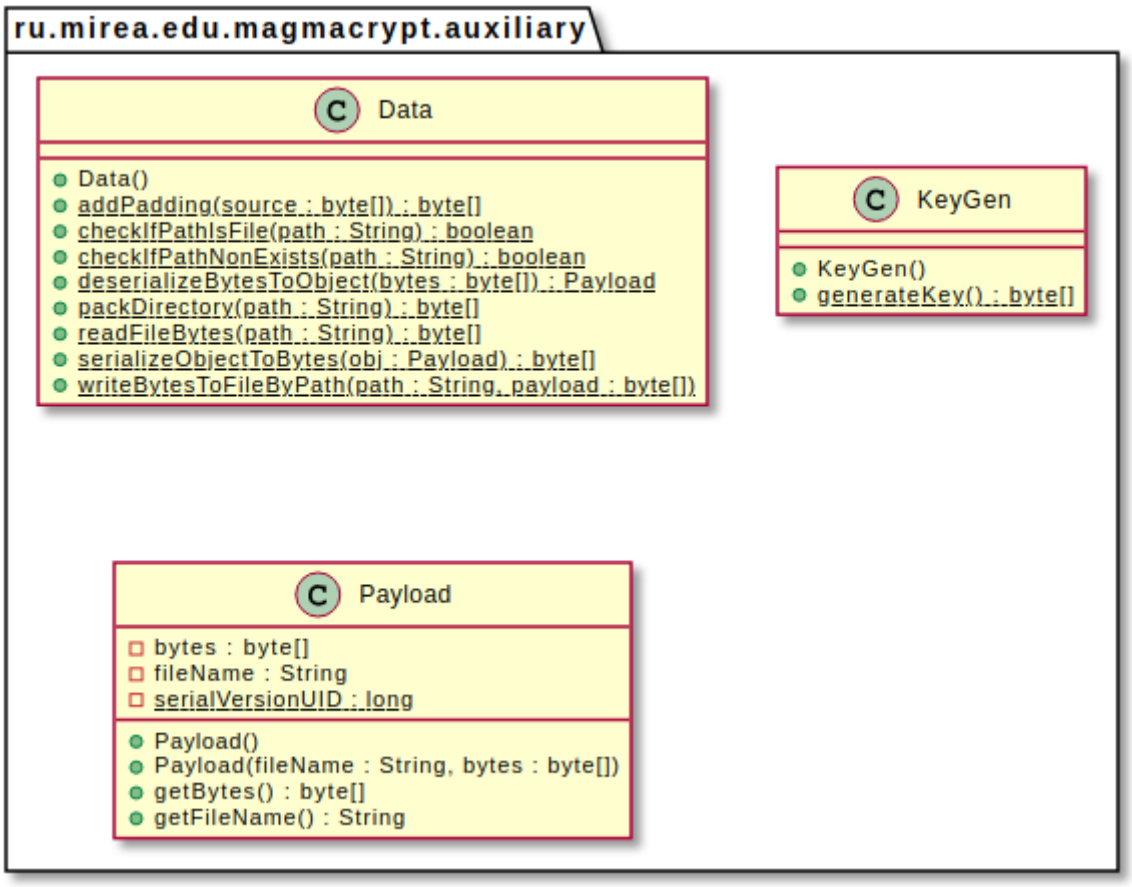

Рис. X: Пакет со вспомогательными сущностями

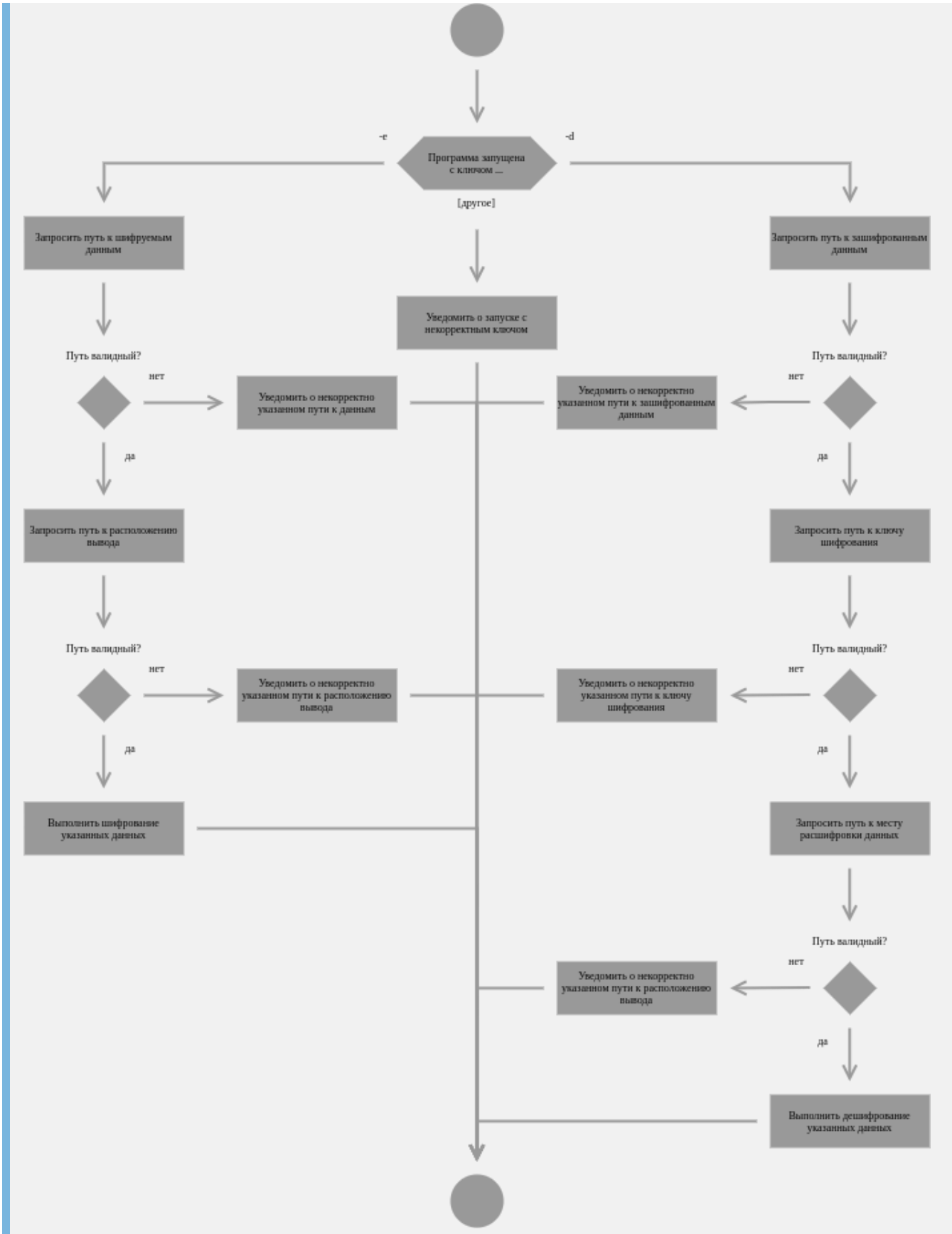

Рис. X: Общая схема работы приложения

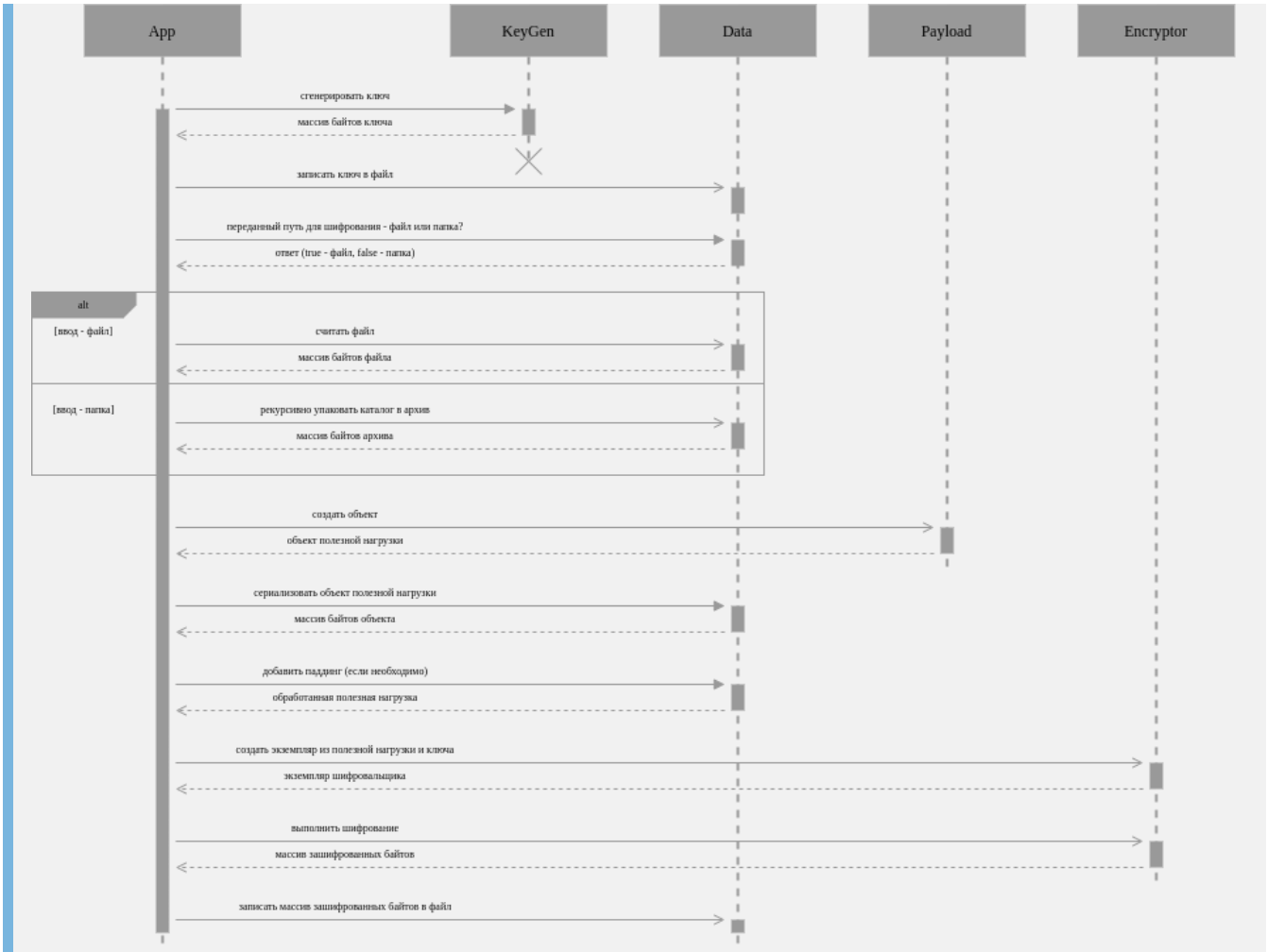

Рис. X: Процесс шифрования

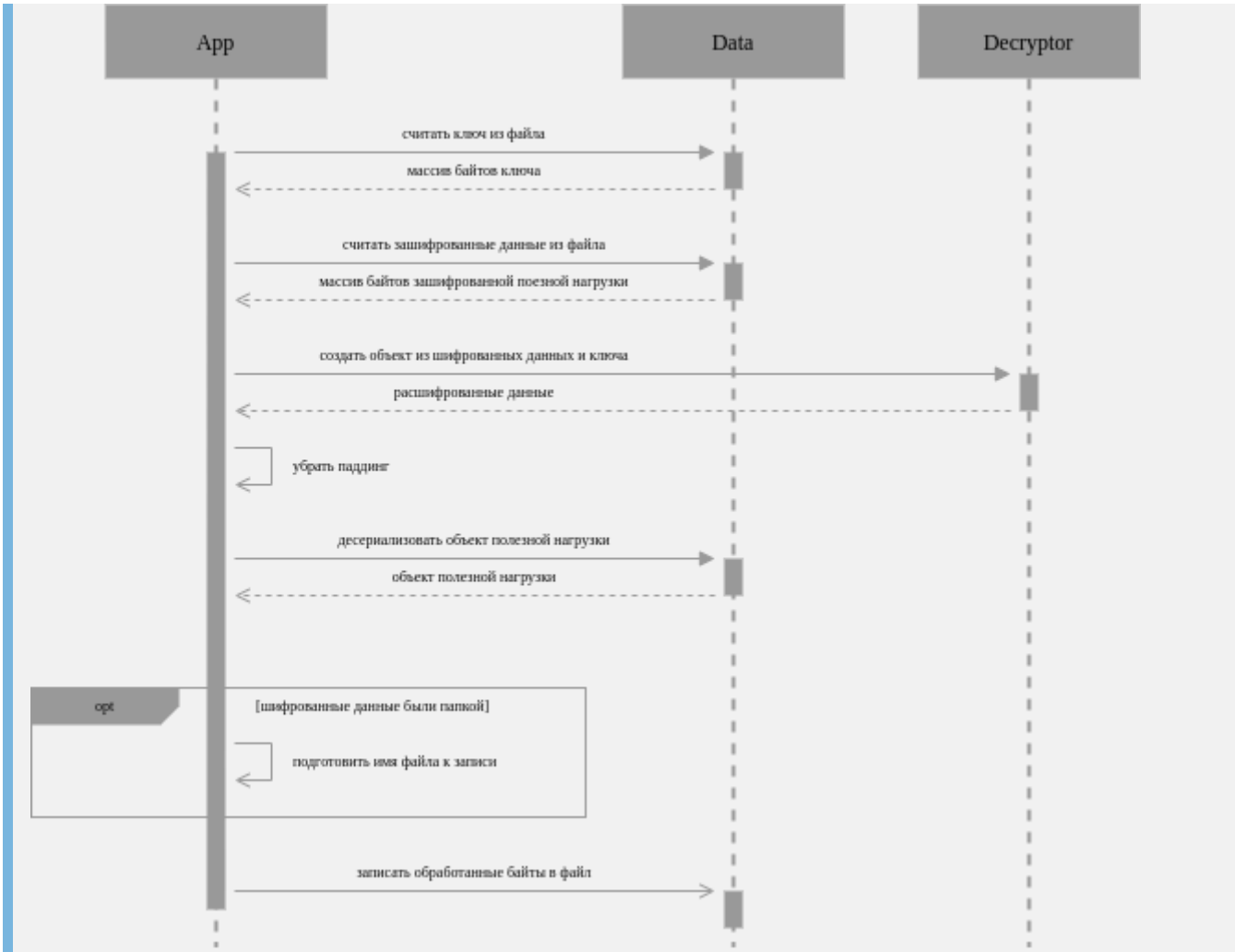

Рис. X: Процесс дешифрования

Методы класса Algorithm:

1. `perform` - абстрактный метод, необходимый для реализации наследниками `Encryptor` и `Decryptor`

```
public abstract byte[] perform();
```

2. `encrypt` - используется классом `Encryptor` в реализации метода `perform`:
    1. Итеративно разделяет входные данные на блоки по 64 бита
    2. Для первых 31 раунда производится шифрование с заменой левой и правой частей блока

```
protected byte[] encrypt(byte[] input, byte[] keySet) {
    byte[] output = Arrays.copyOf(input, input.length);

    for (int i = 0; i <= input.length - 8; i += 8) {
        for (int q = 0; q < 3; q++) {
            for (int w = 0; w < 8; w++) {
                int roundKey = getRoundKey(w, keySet);
                performRound(i, output, roundKey);
            }
```

```
            }
            for (int w = 7; w >= 0; w--) {
                int roundKey = getRoundKey(w, keySet);
                performRound(i, output, roundKey);
                if (w == 0) {
                    byte[] lastArray = new byte[8];
                    System.arraycopy(output, output.length - i - 8,
lastArray, 4, 4);
                    System.arraycopy(output, output.length - i - 4,
lastArray, 0, 4);
                    System.arraycopy(lastArray, 0, output, output.length
- i - 8, 8);
                }
            }
        }

        return output;
    }
```

3. `decrypt` - используется классом `Decryptor` в реализации метода `perform` (процесс аналогичен шифрованию, но порядок раундовых ключей инвертирован):

```
    protected byte[] decrypt(byte[] input, byte[] keySet) {
        byte[] output = Arrays.copyOf(input, input.length);
        int w;

        for (int i = 0; i <= input.length - 8; i += 8) {
            for (w = 0; w < 8; w++) {
                int roundKey = getRoundKey(w, keySet);
                performRound(i, output, roundKey);
            }
            for (int q = 0; q < 3; q++) {
                for (w = 7; w >= 0; w--) {
                    int roundKey = getRoundKey(w, keySet);
                    performRound(i, output, roundKey);
                }
                if (w == -1 && q == 2) {
                    byte[] lastArray = new byte[8];
                    System.arraycopy(output, output.length - i - 8,
lastArray, 4, 4);
                    System.arraycopy(output, output.length - i - 4,
lastArray, 0, 4);
                    System.arraycopy(lastArray, 0, output, output.length
- i - 8, 8);
                }
            }
        }

        return output;
    }
```

4. `performRound` - выполняет изменения в рамках одного раунда шифрования:
    1. выполняет соответствующее преобразование в левой и правых частях текущего блока
    2. выполняет преобразование блока согласно таблице подстановок
    3. смещает получившийся результат на 11 двоичных разрядов

```
 private void performRound(int startIndex, byte[] input, int roundKey)
{
        int n2 = input[input.length - startIndex - 1] << 24
                | (input[input.length - startIndex - 2] & 0xFF) << 16
                | (input[input.length - startIndex - 3] & 0xFF) << 8
                | (input[input.length - startIndex - 4] & 0xFF);
        int n1 = input[input.length - startIndex - 5] << 24
                | (input[input.length - startIndex - 6] & 0xFF) << 16
                | (input[input.length - startIndex - 7] & 0xFF) << 8
                | (input[input.length - startIndex - 8] & 0xFF);
        int s = ((n1 + roundKey) & 0xFFFFFFFF);

        s = substituteUsingSBox(s);
        s = shiftLeftBy11Ranks(s);
        s = s ^ n2;

        byte[] n1s = mergeArrays(n1, s);

        for (int j = 7; j >= 0; j--) {
            input[input.length - 1 - (startIndex + j)] = n1s[j];
        }
 }
```

5. `substituteUsingSBox` - выполняют замену блока согласно таблицы подстановок

```
 private int substituteUsingSBox(int n) {
        int xTest = 0;

        for (int i = 0, j = 0; i <= 28; i += 4, j++) {
            xTest += (sBox[j][(byte) ((n >> i) & 0xF)]) << (i);
        }

        return xTest;
 }
```

6. `mergeArrays` - объединяет левую и правую части блока в рамках 1 раунда

```
 private byte[] mergeArrays(int n1, int n2) {
     byte[] bytes = new byte[8];

     for (int j = 0; j < 4; j++) {
         bytes[j] = (byte) ((n1 >> 24 - (j * 8)) & 0xFF);
     }
```

```
        for (int j = 4; j < 8; j++) {
            bytes[j] = (byte) ((n2 >> 24 - (j * 8)) & 0xFF);
        }

        return bytes;
    }
```

7. `shiftLeftBy11Ranks` - операция сдвига четырехбайтного числа на 11 разрядов

```
    private int shiftLeftBy11Ranks(int a) {
            int shift = 11;
            a = (a >>> (32 - shift)) | a << shift;
            return a;
    }
```

8. `getRoundKey` - получает текущий раундовый ключ по его номеру из исходного ключевой информации

```
    private int getRoundKey(int w, byte[] keySet) {
            return keySet[w * 4 + 3] << 24
                    | (keySet[w * 4 + 2] & 0xFF) << 16
                    | (keySet[w * 4 + 1] & 0xFF) << 8
                    | (keySet[w * 4] & 0xFF);
    }
```

Содержимое пакета `auxiliary` позволяет обеспечить обработку различных способов взаимодействия с данными при проведении процедур шифрования и дешифрования:

1. Класс `Payload` позволяет обернуть и сериализовать в массив байтов байты исходной полезной нагрузки и имя оригинального файла:

```
    public class Payload implements Serializable {
        private static final long serialVersionUID = 1L;
        private final String fileName;
        private final byte[] bytes;

        public Payload(String fileName, byte[] bytes) {
            this.fileName = fileName;
            this.bytes = bytes;
        }

        public Payload() {
            this.fileName = null;
            this.bytes = null;
        }
```

```
    public String getFileName() {
        return this.fileName;
    }

    public byte[] getBytes() {
        return this.bytes;
    }
}
```

2. Класс `KeyGen` содержит статический метод `generateKey` для генерации случайного 256-битного ключа;

3. Класс `Data` предоставляет статические методы для упрощения взаимодействия с файлами:

    1. `packDirectory` - "упаковка" каталог в сжатый zip-архив

```
public static byte[] packDirectory(String path) throws IOException {

    ArrayList<String> entries = new ArrayList<String>();

    Files.walk(Paths.get(path))
            .filter(Files::isRegularFile)
            .forEach(_path -> entries.add(_path.toString()));

    ByteArrayOutputStream out = new ByteArrayOutputStream();
    ZipOutputStream zip = new ZipOutputStream(out);

    for (String entryPath : entries) {
        zip.putNextEntry(new ZipEntry(entryPath));

        BufferedInputStream bufferedInputStream = new
BufferedInputStream(new FileInputStream(new File(entryPath)));
        byte[] bytesInput = new byte[4096];
        int readUnit = 0;

        while ((readUnit = bufferedInputStream.read(bytesInput)) !=
-1) {
            zip.write(bytesInput, 0, readUnit);
        }

        zip.closeEntry();
        bufferedInputStream.close();
    }

    zip.close();
    byte[] output = out.toByteArray();
    out.close();

    return output;
}
```

2. `readFileBytes` - считывает указанный файл в массив байтов

```
public static byte[] readFileBytes(String path) throws IOException {
    return Files.readAllBytes(new File(path).toPath());
}
```

3. `addPadding` - добавляет пустые байты для дополнения последнего блока шифруемой информации

```
public static byte[] addPadding(byte[] source) {
    final int paddingRate = 8 - (source.length % 8);

    byte[] output = new byte[source.length + paddingRate];

    output[0] = (byte) paddingRate;

    for (int i = 0; i < source.length; i++) {
        output[i + 1] = source[i];
    }

    if (paddingRate > 2) {
        output[source.length + 1] = 1;
    }

    return output;
}
```

4. `writeBytesToFileByPath` - записывает байты данных в файл по указанному пути

```
public static void writeBytesToFileByPath(String path, byte[]
payload) throws IOException {
    Path _path = Paths.get(path);
    Files.write(_path, payload);
}
```

5. `checkIfPathIsFile` - проверяет является указанный путь файлом или папкой

```
public static boolean checkIfPathIsFile(String path) {
    File file = new File(path);

    return file.isFile();
}
```

6. `checkIfPathNonExists` - проверяет существование указанного ресурса

```
 public static boolean checkIfPathNonExists(String path) {
     File file = new File(path);

     return Files.notExists(file.toPath());
 }
```

7. `serializeObjectToBytes` - сериализует объект в массив байтов

```
 public static byte[] serializeObjectToBytes(Payload obj) throws
IOException {
     ByteArrayOutputStream boas = new ByteArrayOutputStream();
     ObjectOutputStream ois = new ObjectOutputStream(boas);
     ois.writeObject(obj);
     ois.flush();
     ois.close();
     return boas.toByteArray();
 }
```

8. `deserializeBytesToObject` - восстанавливает объект из массива байтов

```
 public static Payload deserializeBytesToObject(byte[] bytes) throws
IOException, ClassNotFoundException {
     InputStream is = new ByteArrayInputStream(bytes);
     ObjectInputStream ois = new ObjectInputStream(new
ByteArrayInputStream(bytes));
     Payload output = (Payload) ois.readObject();
     ois.close();
     is.close();
     return output;
 }
```

2.3. Пример запуска

При сборке в рамках unit-тестирования выполняются проверки работоспособности отдельно криптоалгоритма, шифрование одного файла и каталога:

```
public class AppTest {
    @Test
    public void algorithmTest() throws NoSuchAlgorithmException {
        final byte[] testKey = KeyGen.generateKey();

        byte[] testPayloadRaw = { 92, 93, -51, 14, -59, -46, 4, 116, 19,
-107, -90, -59, 23, 38, 97, 1, 50, 120, 49,
                -83,
                106, -67, 75, 70, 25, -54, -124, 73, -29, 31, 69, 74, -102,
-47, -27, -35, -60, 50, 86, -38, -125, -64,
                -89, 91, 52, -107, -117, -41, 111, 82, 5, 109, -42, 97, 71,
-2, 2, -113, -51, 78, -28, 121, 8, 84, 54,
                -2, -57, -123, -84, 89 };

        byte[] testPayload = Data.addPadding(testPayloadRaw);

        byte[] enc = new Encryptor(testPayload, testKey).perform();
        byte[] dec = new Decryptor(enc, testKey).perform();

        byte[] cut = Arrays.copyOfRange(dec, 1, dec.length - dec[0] + 1);
        assertTrue("Algorithm work check pass: ",
Arrays.equals(testPayloadRaw, cut));
    }

    @Test
    public void generalTest() throws NoSuchAlgorithmException, IOException,
ClassNotFoundException {

        final String TEST_DATA_RESOURCES_PATH = "./TEST_DATA/RESOURCES/";
        final String TEST_DATA_OUTPUT_PATH = "./TEST_DATA/OUTPUT/";

        // ? - SINGLE FILE TEST
        encrypt(TEST_DATA_RESOURCES_PATH + "SINGLE_BINARY_FILE.jpeg",
TEST_DATA_OUTPUT_PATH);
        decrypt(TEST_DATA_OUTPUT_PATH + "ENCRYPTED", TEST_DATA_OUTPUT_PATH
+ "KEY", TEST_DATA_OUTPUT_PATH);

        byte[] decryptedZipBytes = Data.readFileBytes(TEST_DATA_OUTPUT_PATH
+ "SINGLE_BINARY_FILE.jpeg");

        byte[] testPayload = Data.readFileBytes(TEST_DATA_OUTPUT_PATH +
"SINGLE_BINARY_FILE.jpeg");

        boolean fileTestResult = Arrays.equals(decryptedZipBytes,
testPayload);;
```

```
        // ? - DIRECTORY TEST
        encrypt(TEST_DATA_RESOURCES_PATH, TEST_DATA_OUTPUT_PATH);
        decrypt(TEST_DATA_OUTPUT_PATH + "ENCRYPTED", TEST_DATA_OUTPUT_PATH
+ "KEY", TEST_DATA_OUTPUT_PATH);

        decryptedZipBytes = Data.readFileBytes(TEST_DATA_OUTPUT_PATH +
"RESTORED.ZIP");

        testPayload = Data.packDirectory(TEST_DATA_RESOURCES_PATH);

        boolean dirTestResult = Arrays.equals(decryptedZipBytes,
testPayload);

        assertTrue("General application check pass: ", dirTestResult &&
fileTestResult);
    }
}
```

При ручном запуске данные сценарии могут быть воспроизведены следующим образом:

1. Шифрование единичного файла

```
$ java -jar ./target/magmacrypt-1.0.jar
Specify path to data: ./TEST_DATA/RESOURCES/SINGLE_SOURCE_CODE.java
Where place encrypted data: ./TEST_DATA/OUTPUT/

$ tree TEST_DATA
TEST_DATA
├── OUTPUT
│   ├── DO_NOT_DELETE
│   ├── ENCRYPTED
│   └── KEY
└── RESOURCES
    ├── SINGLE_BINARY_FILE.jpeg
    └── SINGLE_SOURCE_CODE.java

2 directories, 5 files

$ java -jar ./target/magmacrypt-1.0.jar -d
Specify path to encrypted file: ./TEST_DATA/OUTPUT/ENCRYPTED
Specify path to key file: ./TEST_DATA/OUTPUT/KEY
Where place decrypted data: ./TEST_DATA/OUTPUT/

TEST_DATA
├── OUTPUT
│   ├── DO_NOT_DELETE
│   ├── ENCRYPTED
│   ├── KEY
│   └── SINGLE_SOURCE_CODE.java
└── RESOURCES
    ├── SINGLE_BINARY_FILE.jpeg
    └── SINGLE_SOURCE_CODE.java
```

```
2 directories, 6 files

$ cat TEST_DATA/RESOURCES/SINGLE_SOURCE_CODE.java && echo '\n' && cat
TEST_DATA/OUTPUT/SINGLE_SOURCE_CODE.java
public class MyClass {
  int x = 5;

  public static void main(String[] args) {
    MyClass myObj1 = new MyClass();  // Object 1
    MyClass myObj2 = new MyClass();  // Object 2
    System.out.println(myObj1.x);
    System.out.println(myObj2.x);
  }
}

public class MyClass {
  int x = 5;

  public static void main(String[] args) {
    MyClass myObj1 = new MyClass();  // Object 1
    MyClass myObj2 = new MyClass();  // Object 2
    System.out.println(myObj1.x);
    System.out.println(myObj2.x);
}
```

2. Шифрование каталога

```
$ java -jar ./target/magmacrypt-1.0.jar
Specify path to data: ./TEST_DATA/RESOURCES/
Where place encrypted data: ./TEST_DATA/OUTPUT/

$ java -jar ./target/magmacrypt-1.0.jar -d
Specify path to encrypted file: ./TEST_DATA/OUTPUT/ENCRYPTED
Specify path to key file: ./TEST_DATA/OUTPUT/KEY
Where place decrypted data: ./TEST_DATA/OUTPUT/

$ unzip -l TEST_DATA/OUTPUT/RESTORED.ZIP
Archive:  TEST_DATA/OUTPUT/RESTORED.ZIP
  Length      Date    Time    Name
---------  ---------- -----   ----
   334137  2021-12-25 05:13   ./TEST_DATA/RESOURCES/SINGLE_BINARY_FILE.jpeg
      251  2021-12-25 05:13   ./TEST_DATA/RESOURCES/SINGLE_SOURCE_CODE.java
---------                     -------
   334388                     2 files
```

Заключение

В данной работе я реализовала алгоритм шифрования ГОСТ 28147–89, а также проверила результат работы на различных типах полезной нагрузки. Шифр является устойчивым к атакам путём полного перебора.