Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

КАФЕДРА ИНФОРМАТИКИ

Лабораторная работа № 1

**Компьютерная реализация блочных шифров на примере DES и ГОСТ**

Выполнил студент гр. 953501

Трофимук Г.А

Проверил

Протько М.И.

Минск, 2022

# 1. Введение

Реализовать программные средства шифрования и дешифрования текстовых файлов при помощи алгоритмов DES и ГОСТ 28147-89

**2. Теоретические сведения**

**Общая схема DES**

DES является классической сетью Фейстеля с двумя ветвями.

Данные шифруются 64-битными блоками, используя 56-битный ключ.

Алгоритм преобразует за несколько раундов 64-битный вход в 64-битный выход.

Процесс шифрования состоит из четырех этапов.

* Выполняется начальная перестановка (IP) 64-битного исходного текста (забеливание), во время которой биты переупорядочиваются в соответствии со стандартной таблицей.
* Этап состоит из 16 раундов одной и той же функции, которая использует операции сдвига и подстановки.
* Левая и правая половины выхода последней (16-й) итерации меняются местами.
* Выполняется перестановка IP-1 результата, полученного на третьем этапе.

Перестановка IP-1 инверсна начальной перестановке.



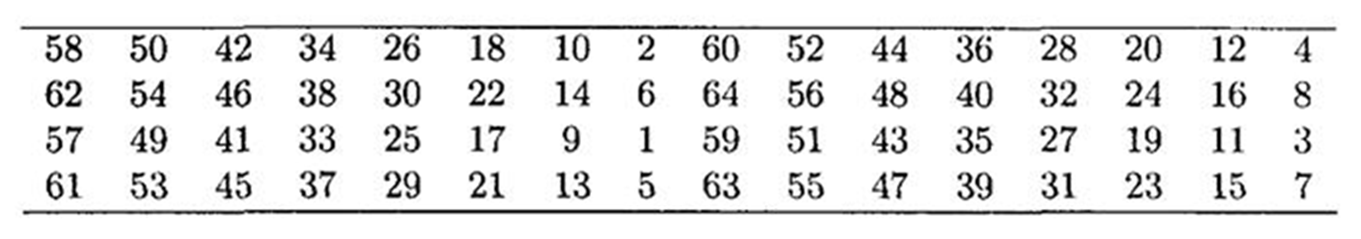
**Шифрование**

**Начальная перестановка**

Начальная перестановка и ее инверсия определяются стандартной таблицей.

Если **М** - это произвольные 64 бита, то

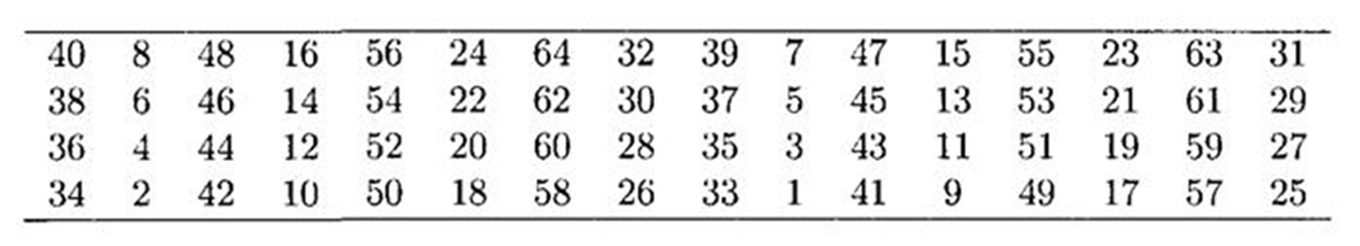
**X = IP (M)** - переставленные 64 бита.



Если применить обратную функцию перестановки

**Y = IP-1 (X) = IP-1 (IP(M)),**

то получится первоначальная последовательность бит.



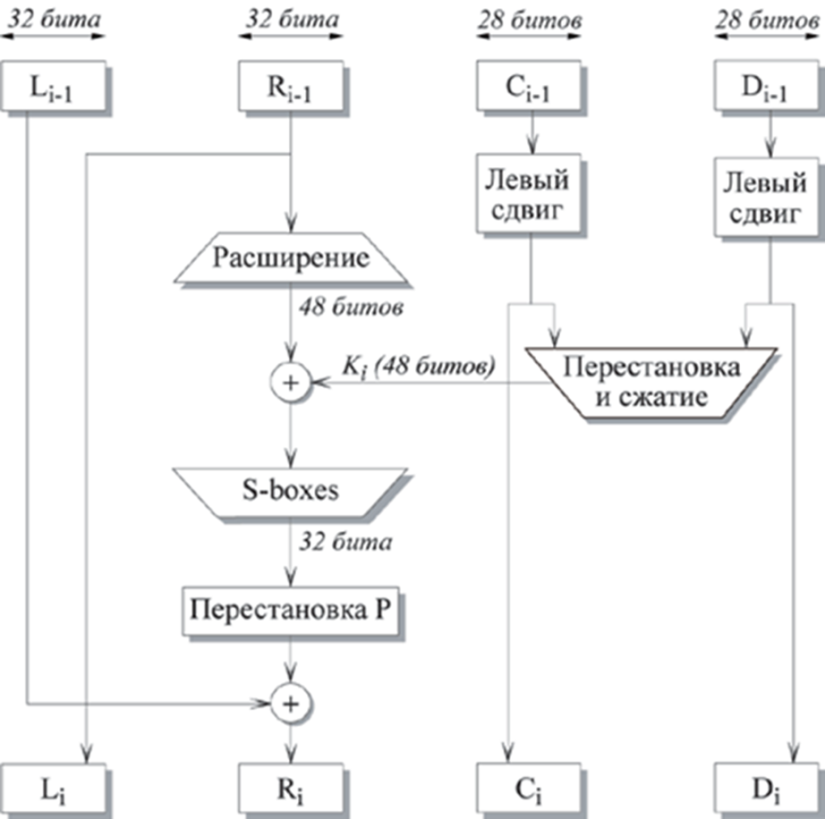
Начальная перестановка и соответствующая заключительная перестановка не влияют на криптостойкость DES (перестановка в первую очередь служит для облегчения побайтной загрузки данных открытого текста и шифротекста в микросхему DES).

**Последовательность преобразований отдельного раунда**

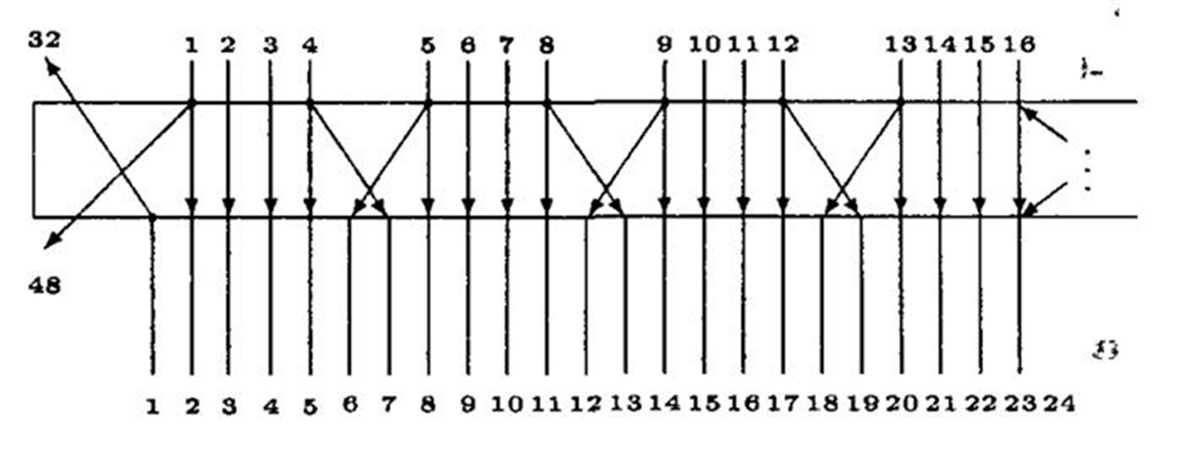
Каждую итерацию можно описать следующим образом:

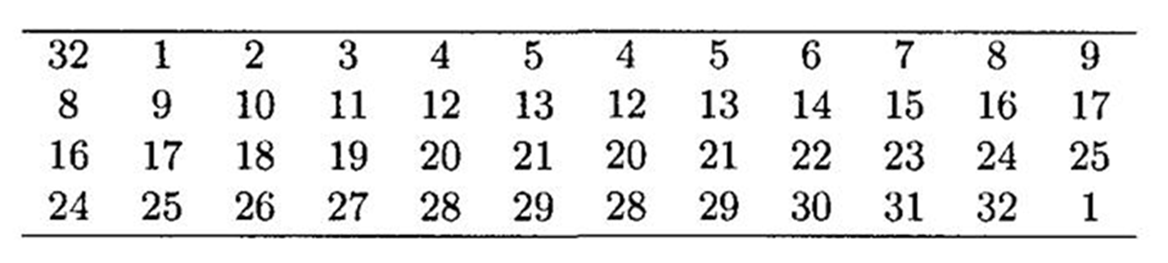
Li = Ri-1

Ri = Li-1⊕F(Ri-1, Ki)



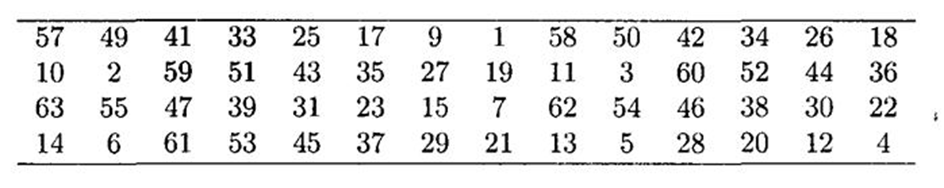
**DES — схема перестановки с расширением**



****

**Процедура преобразования ключа**

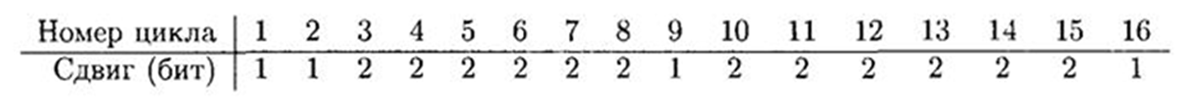
64-битовый ключ DES уменьшается до 56-битового ключа отбрасыванием каждого восьмого бита. Эти биты используются только для контроля четности, позволяя проверять правильность ключа.



После извлечения 56-битного ключа для каждого из 16 циклов генерируется новый 48-битный подключ.

Эти подключи, К2, определяются следующим образом:

* 56-битный ключ делится на две 28-битные половины.
* Половины циклически сдвигаются влево на один или два бита в зависимости от номера цикла.



**DES — подстановка при помощи S-блоков**

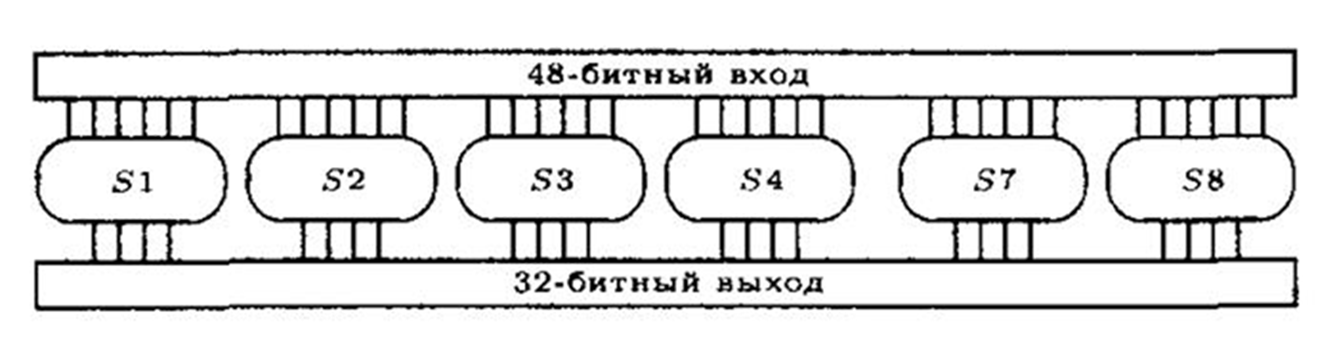
После объединения сжатого блока с расширенным блоком с помощью XOR над 48-битным результатом выполняется операция подстановки.

Подстановки производятся в восьми блоках подстановки, или S-блоках (от substitution).

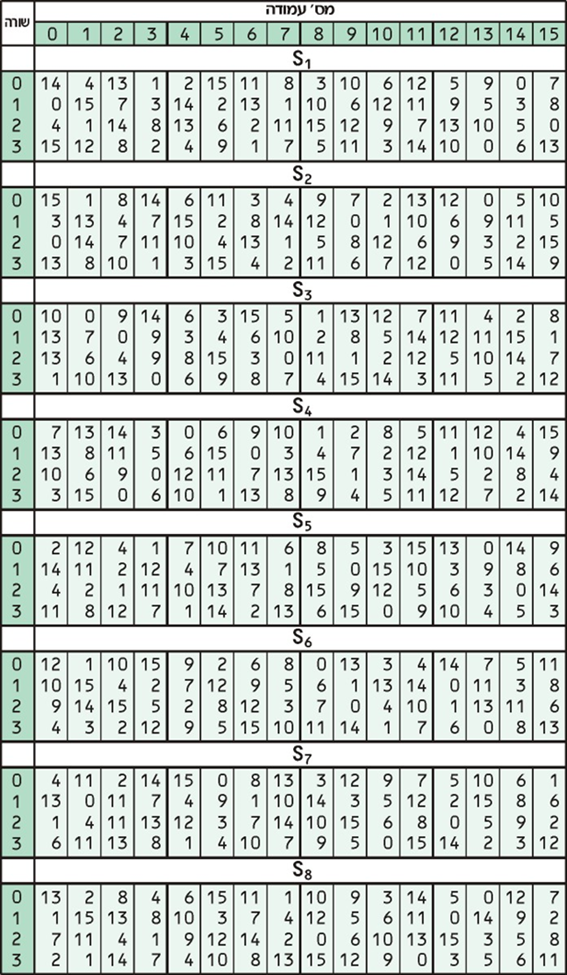
У каждого S-блока есть 6-битный вход и 4-битный выход, всего используется восемь различных S-блоков.

48 битов делятся на восемь 6-битных подблоков.

Каждый отдельный подблок обрабатывается отдельным S-блоком: первый подблок - первым S-блоком, второй - вторым S-блоком и так далее.



Каждый S-блок представляет собой таблицу из четырех строк и шестнадцати столбцов. Каждый элемент в блоке является 4-битным числом. По шести входным битам S-блока определяется, под какими номерами столбцов и строк следует искать выходное значение.



Например, пусть на вход шестого S-блока (то есть биты функции XOR с 31 по 36) попадает 110011. Первый и последний биты, объединяясь, образуют 11, что соответствует строке три шестого S-блока. Средние четыре бита образуют 1001, что соответствует столбцу девять того же S-блока. Элемент шестого S-блока, находящийся на пересечении строки три и столбца девять, - это 14 (строки и столбцы нумеруются с нуля, а не с единицы). Вместо 110011 подставляется 1110.

**Дешифрование**

Процесс дешифрования аналогичен процессу шифрования.

На входе алгоритма используется зашифрованный текст, но ключи *Ki* используются в обратной последовательности.

*K16* используется на первом *раунде*, *K1* используется на последнем *раунде*.

**Проблемой DES является малая длина ключа.**

Также без ответа пока остается вопрос, возможен ли криптоанализ с использованием существующих характеристик алгоритма DES.

Основой алгоритма являются восемь таблиц подстановки, или S-boxes, которые применяются в каждой итерации.

Существует опасность, что эти S-boxes конструировались таким образом, что криптоанализ возможен для взломщика, который знает слабые места S-boxes.

В течение многих лет обсуждалось как стандартное, так и неожиданное поведение S-boxes, но все-таки никому не удалось обнаружить их фатально слабые места.

**Двойной DES**

Простейший способ увеличить длину ключа состоит в повторном применении *DES* с двумя разными ключами.

Используя незашифрованное сообщение P и два ключа K1 и K2, зашифрованное сообщение С можно получить следующим образом:

**C = Ek2 [Ek1 [P]]**

При дешифрования два ключа применяются в обратном порядке:

**P = Dk1 [Dk2 [C]]**

В этом случае длина ключа равна 56 \* 2 = 112 бит.

**Атака "встреча посередине"**

Для приведенного выше алгоритма двойного *DES* существует так называемая атака "встреча посередине".

Она основана на следующем свойстве алгоритма. Мы имеем

С = Ek2 [Ek1 [P]]

Тогда

X = Ek1 [P] = Dk2 [C].

Атака состоит в следующем.

* Требуется, чтобы атакующий знал хотя бы одну пару незашифрованный текст и соответствующий ему зашифрованный текст: (Р, С).
* Шифруется Р для всех возможных 256 значений K1.
* Результат запоминается в таблице, и затем таблица упорядочивается по значению Х.
* Дешифруется С, с применением всех возможных 256 значений K2.
* Для каждого выполненного дешифрования ищется равное ему значение в первой таблице.
* Если соответствующее значение найдено, то считается, что эти ключи могут быть правильными, и они проверяются для следующей известной пары незашифрованный текст, зашифрованный текст.

**Тройной DES с двумя ключами**

Очевидное противодействие атаке "встреча посередине" состоит в использовании третьей стадии шифрования с тремя различными ключами.

Это поднимает стоимость лобовой атаки до 2168, которая на сегодняшний день считается выше практических возможностей.

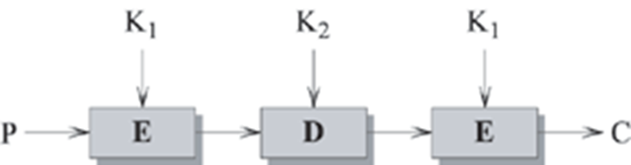
Но при этом длина ключа равна 56 \* 3 = 168 бит, что иногда бывает громоздко.

В качестве альтернативы предлагается метод тройного шифрования, использующий только два ключа.

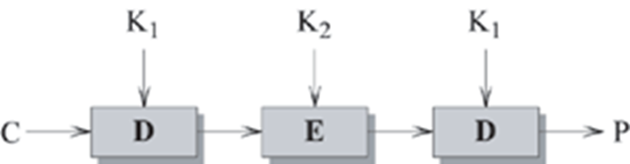
В этом случае выполняется последовательность зашифрование-расшифрование-зашифрование (EDE).

C = EK1 [DK2 [EK1 [P]]]

Шифрование тройным DES



Дешифрование тройным DES



Не имеет большого значения, что используется на второй стадии: шифрование или дешифрование.

В случае использования дешифрования существует только то преимущество, что можно *тройной DES* свести к обычному одиночному *DES*, используя K1 = K2:

**C = EK1 [DK1 [EK1 [P]]] = EK1 [P]**

Известных криптографических атак на *тройной DES* не существует.

Цена подбора ключа в *тройном DES* равна **2112**.

## Стандарт шифрования ГОСТ 28147-89

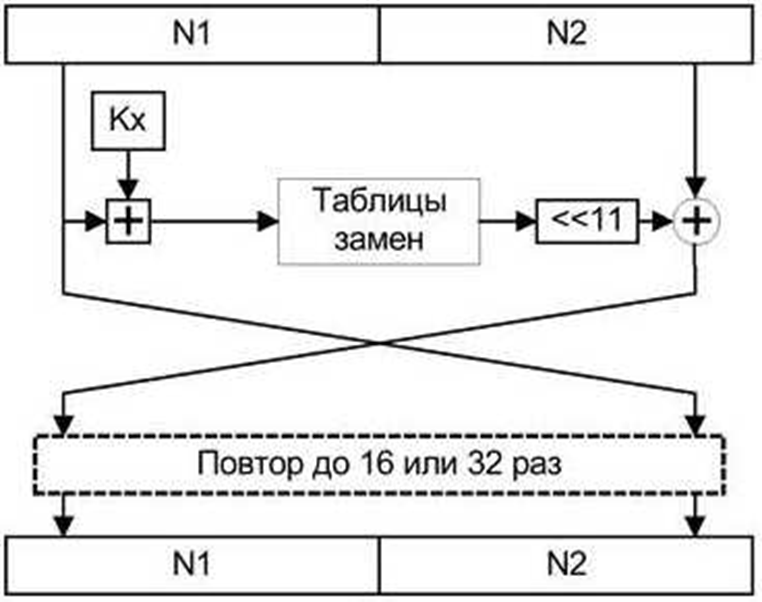
**ГОСТ 28147-89** представляет собой симметричный 64-битовый блочный алгоритм с 256-битовым ключом.

Этот алгоритм криптографического преобразования данных предназначен для аппаратной и программной реализации, удовлетворяет криптографическим требованиям и не накладывает ограничений на степень секретности защищаемой информации.

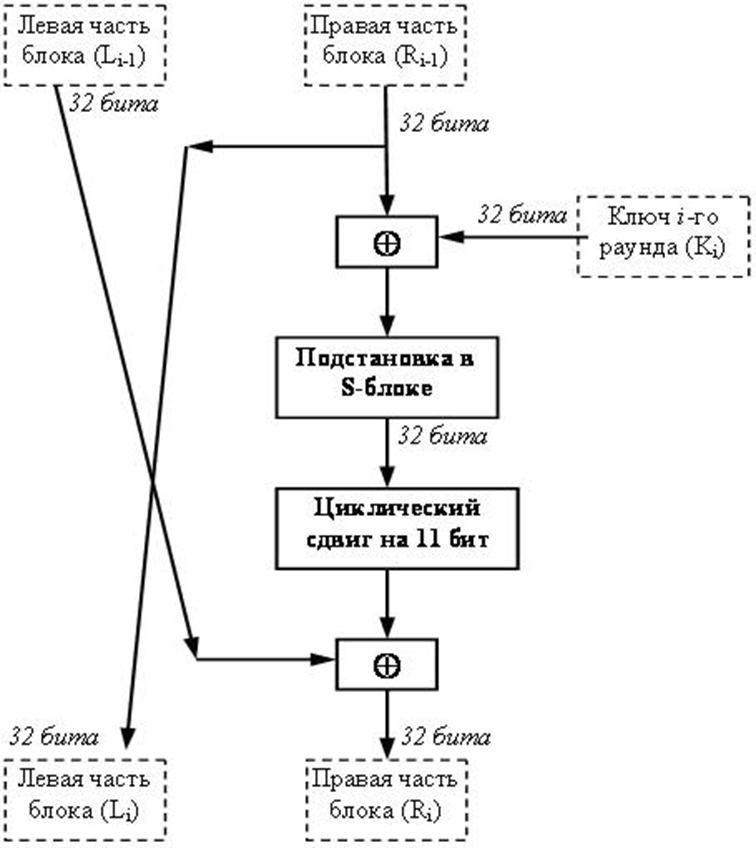
**Схема алгоритма ГОСТ 28147-89**

Данные, подлежащие зашифровке, разбивают на 64-разрядные блоки.

Эти блоки разбиваются на два субблока N1 и N2 по 32 бит



**Структура одного раунда ГОСТ 28147-89**



**Описание работы ГОСТ 28147-89**

Шифруемый блок данных разбивается на две части, которые затем обрабатываются как отдельные 32-битовые целые числа без знака.

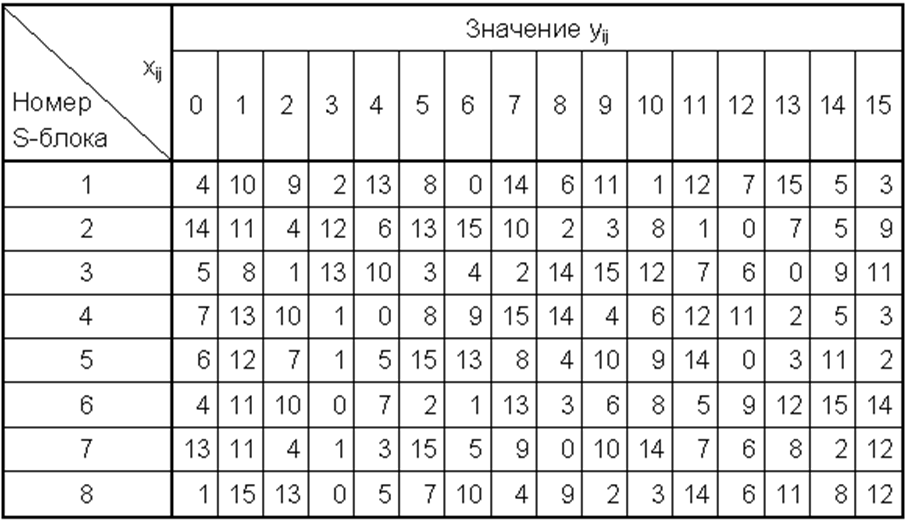
Сначала правая половина блока и подключ раунда складываются по модулю 232.

Затем производится поблочная подстановка.

32-битовое значение, полученное на предыдущем шаге (обозначим его S), интерпретируется как массив из восьми 4-битовых блоков кода: S=(S0,S1,S2,S3,S4,S5,S6,S7).

Далее значение каждого из восьми блоков заменяется на новое, которое выбирается **по таблице замен**

Таблица замен ГОСТ 28147-89



В каждой строке таблицы замен записаны числа от 0 до 15 в произвольном порядке без повторений.

Значения элементов таблицы замен взяты от 0 до 15, так как в четырех битах, которые подвергаются подстановке, может быть записано целое число без знака в диапазоне от 0 до 15.

Значение блока S1 (четыре младших бита 32-разрядного числа S) заменится на число, стоящее на позиции, номер которой равен значению заменяемого блока.

Например, в этом случае S1=0 заменится на 4, если S1=1, то оно заменится на 10 и т.д.

**Описание ГОСТ 28147-89**

После выполнения подстановки все 4-битовые блоки снова объединяются в единое 32-битное слово, которое затем циклически сдвигается на 11 битов влево.

Наконец, с помощью побитовой операции "сумма по модулю 2" результат объединяется с левой половиной, вследствие чего получается новая правая половина Ri.

Новая левая часть Li берется равной младшей части преобразуемого блока: Li= Ri-1.

Полученное значение преобразуемого блока рассматривается как результат выполнения одного раунда алгоритма шифрования.

**Процедуры шифрования и расшифрования**

ГОСТ 28147-89 является блочным шифром, поэтому преобразование данных осуществляется блоками в так называемых базовых циклах.

Базовые циклы заключаются в многократном выполнении для блока данных основного раунда, рассмотренного нами ранее, с использованием разных элементов ключа и отличаются друг от друга порядком использования ключевых элементов.

В каждом раунде используется один из восьми возможных 32-разрядных подключей.

Рассмотрим процесс создания подключей раундов.  
В ГОСТ эта процедура очень проста, особенно по сравнению с DES.  
256-битный ключ K разбивается на восемь 32-битных подключей, обозначаемых K0, K1, K2, K3, K4, K5, K6, K7.   
Алгоритм включает 32 раунда, поэтому каждый подключ при шифровании используется в четырех раундах в последовательности, представленной в таблице

Таблица Последовательность использования подключей при шифровании

Раунд 1 2 3 4 5 6 7 8

Подключ K0 K1 K2 K3 K4 K5 K6 K7

Раунд 9 10 11 12 13 14 15 16

Подключ K0 K1 K2 K3 K4 K5 K6 K7

Раунд 17 18 19 20 21 22 23 24

Подключ K0 K1 K2 K3 K4 K5 K6 K7

Раунд 25 26 27 28 29 30 31 32

Подключ K7 K6 K5 K4 K3 K2 K1 K0

Процесс расшифрования производится по тому же алгоритму, что и шифрование. Единственное отличие заключается в порядке использования подключей Ki. При расшифровании подключи должны быть использованы в обратном порядке, а именно, как указано в таблице

Таблица Последовательность использования подключей при расшифровании

Раунд 1 2 3 4 5 6 7 8

Подключ K0 K1 K2 K3 K4 K5 K6 K7

Раунд 9 10 11 12 13 14 15 16

Подключ K7 K6 K5 K4 K3 K2 K1 K0

Раунд 17 18 19 20 21 22 23 24

Подключ K7 K6 K5 K4 K3 K2 K1 K0

Раунд 25 26 27 28 29 30 31 32

Подключ K7 K6 K5 K4 K3 K2 K1 K0

**ГОСТ 28147-89 Режимы работы**

**Режим простой замены**: все блоки шифруются независимо друг от друга с разными подключами в разных раундах. Для одинаковых блоков сообщения М блоки шифртекста будут одинаковыми.

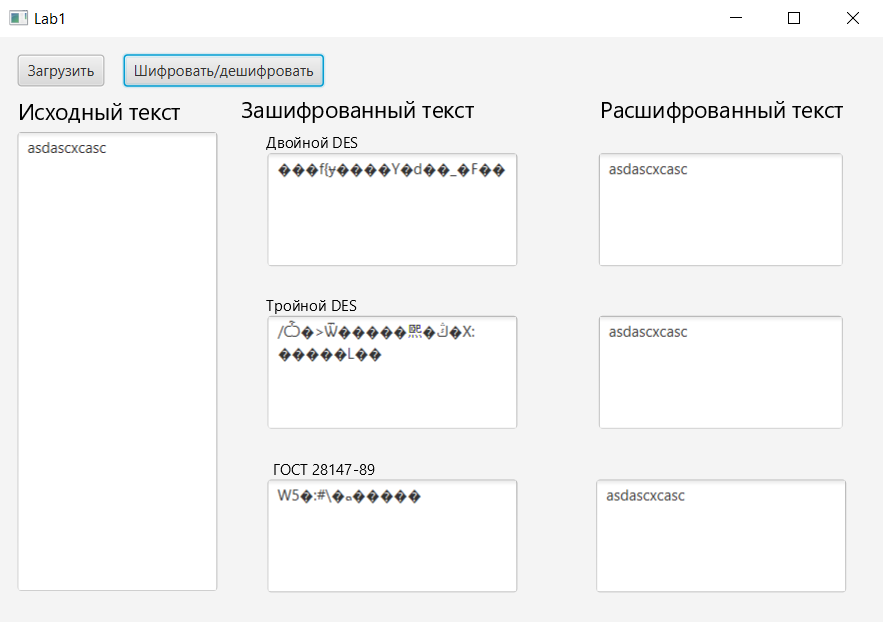
**Режим гаммирования**: В регистры N1 и N2 записывается 64-битовая синхропосылка (вектор инициализации) и шифруется с использованием СК. Результат подается на вход регистров и снова шифруется с использованием ключа. Получается «одноразовый блокнот».

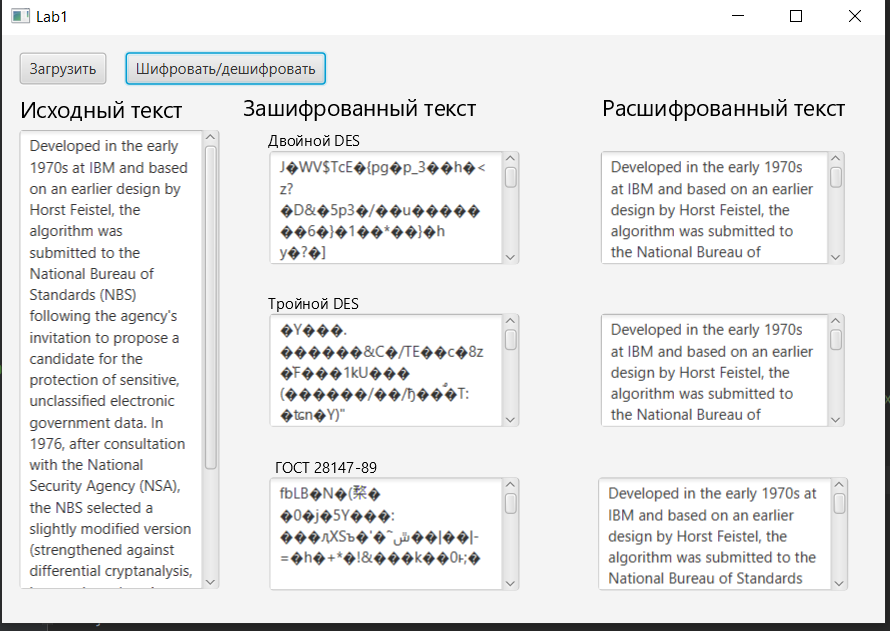
**В режиме гаммирования с обратной связью** для заполнения регистров N1 и N2, начиная со 2-го блока, используется результат зашифрования предыдущего блока открытого текста.

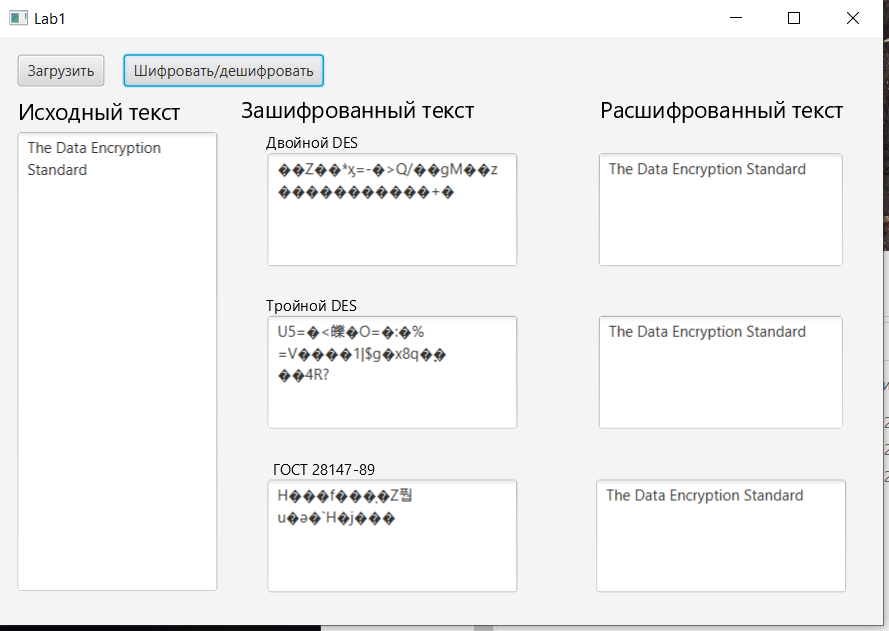
**Работа криптосистемы в режиме гаммирования**

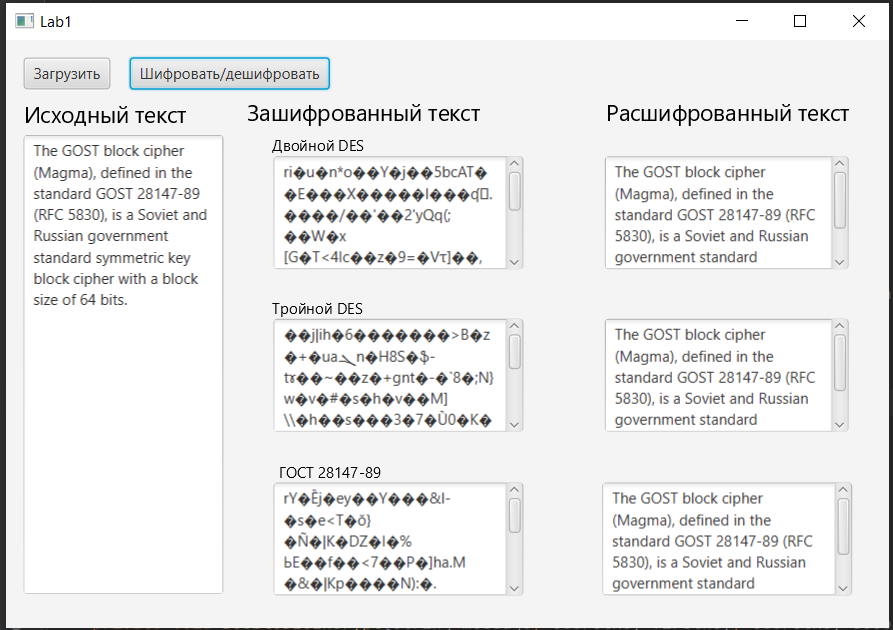


# 4. Результаты выполнения программы









# 5. Выводы

DES является классической сетью Фейстеля с двумя ветвями. Данные шифруются 64-битными блоками, используя 56-битный ключ. Проблемой DES является малая длина ключа. Простейший способ увеличить длину ключа состоит в повторном применении DES с двумя разными ключами. Однако двойной DES может быть подвергнут атаке "встреча посередине". Очевидное противодействие атаке "встреча посередине" состоит в использовании третьей стадии шифрования с тремя различными ключами, то есть тройной DES. Это поднимает стоимость лобовой атаки до 2168, которая на сегодняшний день считается выше практических возможностей. Известных криптографических атак на тройной DES не существует.

ГОСТ 28147-89 представляет собой симметричный 64-битовый блочный алгоритм с 256-битовым ключом. Этот алгоритм криптографического преобразования данных предназначен для аппаратной и программной реализации, удовлетворяет криптографическим требованиям и не накладывает ограничений на степень секретности защищаемой информации.

# 6. КОД ПРОГРАММЫ

**Main.java:**

package com.example.lab1si;  
  
import javafx.application.Application;  
import javafx.fxml.FXMLLoader;  
import javafx.scene.Scene;  
import javafx.stage.Stage;  
  
import java.io.IOException;  
import java.security.Security;  
import org.bouncycastle.jce.provider.BouncyCastleProvider;  
  
public class Main extends Application {  
 @Override  
 public void start(Stage stage) throws IOException {  
 FXMLLoader fxmlLoader = new FXMLLoader(Main.class.getResource("view.fxml"));  
 Scene scene = new Scene(fxmlLoader.load());  
 stage.setTitle("Lab1");  
 stage.setScene(scene);  
 stage.show();  
 Controller controller = fxmlLoader.getController();  
 controller.setStage(stage);  
 Security.*addProvider*(new BouncyCastleProvider());  
 }  
  
 public static void main(String[] args) {  
 *launch*();  
 }  
}

**Controller.java:**

package com.example.lab1si;  
  
import javafx.event.ActionEvent;  
import javafx.fxml.FXML;  
import javafx.scene.control.TextArea;  
import javafx.stage.FileChooser;  
import javafx.stage.Stage;  
  
import javax.crypto.\*;  
import java.io.File;  
import java.io.FileReader;  
import java.io.IOException;  
import java.nio.charset.StandardCharsets;  
import java.security.InvalidKeyException;  
import java.security.NoSuchAlgorithmException;  
import java.security.NoSuchProviderException;  
  
  
public class Controller {  
 private Stage stage;  
 @FXML  
 private TextArea sourceTextArea;  
 @FXML  
 private TextArea encryptDoubleDESTextArea;  
 @FXML  
 private TextArea decryptDoubleDESTextArea;  
 @FXML  
 private TextArea encryptTripleDESTextArea;  
 @FXML  
 private TextArea decryptTripleDESTextArea;  
 @FXML  
 private TextArea encryptGOSTTextArea;  
 @FXML  
 private TextArea decryptGOSTTextArea;  
  
 public Stage getStage() {  
 return stage;  
 }  
  
 public void setStage(Stage stage) {  
 this.stage = stage;  
 }  
  
 @FXML  
 void Load(ActionEvent event) throws IOException {  
 final FileChooser fileChooser = new FileChooser();  
 fileChooser.setInitialDirectory(new File("C:\\Users\\User\\Desktop\\Tests\\Txts"));  
 File file = fileChooser.showOpenDialog(stage);  
 if (file != null) {  
 FileReader reader = new FileReader(file.getAbsolutePath());  
 StringBuilder result = new StringBuilder();  
 int character;  
 while((character=reader.read())!=-1)  
 {  
 result.append((char) character);  
 }  
 sourceTextArea.setText(result.toString());  
 }  
 }  
 @FXML  
 void EncryptDecrypt(ActionEvent event) throws NoSuchAlgorithmException, NoSuchPaddingException, IllegalBlockSizeException, BadPaddingException, InvalidKeyException, NoSuchProviderException {  
 KeyGenerator keyGenerator = KeyGenerator.*getInstance*("DES");  
 SecretKey firstDESKey = keyGenerator.generateKey();  
 SecretKey secondDESKey = keyGenerator.generateKey();  
 keyGenerator = KeyGenerator.*getInstance*("GOST28147");  
 SecretKey keyGOST = keyGenerator.generateKey();  
  
 DoubleDESEncrypt(sourceTextArea.getText(),firstDESKey,secondDESKey);  
 TripleDESEncrypt(sourceTextArea.getText(),firstDESKey,secondDESKey);  
 GOSTEncrypt(sourceTextArea.getText(),keyGOST);  
 }  
 private void DoubleDESEncrypt(String message, SecretKey firstKey, SecretKey secondKey) throws NoSuchPaddingException, NoSuchAlgorithmException, InvalidKeyException, IllegalBlockSizeException, BadPaddingException {  
 Cipher cipher = Cipher.*getInstance*("DES");  
 cipher.init(Cipher.*ENCRYPT\_MODE*, firstKey);  
 byte[] resultMessage = cipher.doFinal(message.getBytes());  
 cipher.init(Cipher.*ENCRYPT\_MODE*, secondKey);  
 resultMessage = cipher.doFinal(resultMessage);  
 encryptDoubleDESTextArea.setText(new String(resultMessage, StandardCharsets.*UTF\_8*));  
 cipher.init(Cipher.*DECRYPT\_MODE*, secondKey);  
 resultMessage = cipher.doFinal(resultMessage);  
 cipher.init(Cipher.*DECRYPT\_MODE*, firstKey);  
 resultMessage = cipher.doFinal(resultMessage);  
 decryptDoubleDESTextArea.setText(new String(resultMessage, StandardCharsets.*UTF\_8*));  
 }  
 private void TripleDESEncrypt(String message, SecretKey firstKey, SecretKey secondKey) throws NoSuchPaddingException, NoSuchAlgorithmException, InvalidKeyException, IllegalBlockSizeException, BadPaddingException {  
 Cipher cipher = Cipher.*getInstance*("DES");  
 cipher.init(Cipher.*ENCRYPT\_MODE*, firstKey);  
 byte[] resultMessage = cipher.doFinal(message.getBytes());  
 cipher.init(Cipher.*ENCRYPT\_MODE*, secondKey);  
 resultMessage = cipher.doFinal(resultMessage);  
 cipher.init(Cipher.*ENCRYPT\_MODE*, firstKey);  
 resultMessage = cipher.doFinal(resultMessage);  
 encryptTripleDESTextArea.setText(new String(resultMessage, StandardCharsets.*UTF\_8*));  
 cipher.init(Cipher.*DECRYPT\_MODE*, firstKey);  
 resultMessage = cipher.doFinal(resultMessage);  
 cipher.init(Cipher.*DECRYPT\_MODE*, secondKey);  
 resultMessage = cipher.doFinal(resultMessage);  
 cipher.init(Cipher.*DECRYPT\_MODE*, firstKey);  
 resultMessage = cipher.doFinal(resultMessage);  
 decryptTripleDESTextArea.setText(new String(resultMessage, StandardCharsets.*UTF\_8*));  
 }  
 private void GOSTEncrypt(String message, SecretKey key) throws NoSuchPaddingException, NoSuchAlgorithmException, InvalidKeyException, NoSuchProviderException, IllegalBlockSizeException, BadPaddingException {  
 Cipher cipher = Cipher.*getInstance*("GOST28147/ECB/PKCS5Padding", "BC");  
 cipher.init(Cipher.*ENCRYPT\_MODE*, key);  
 byte[] resultMessage = cipher.doFinal(message.getBytes());  
 encryptGOSTTextArea.setText(new String(resultMessage, StandardCharsets.*UTF\_8*));  
 cipher.init(Cipher.*DECRYPT\_MODE*, key);  
 resultMessage = cipher.doFinal(resultMessage);  
 decryptGOSTTextArea.setText(new String(resultMessage, StandardCharsets.*UTF\_8*));  
 }  
}