Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования

«Рязанский государственный радиотехнический

университет имени В.Ф. Уткина»

Кафедра «ЭВМ»

Отчет о лабораторной работе №3

«Поточные шифры»

по дисциплине

«Защита информации»

Выполнили:

Студенты группы 045

Анохин В.А.

Вашкулатов Н.А.

Проверили:

доц. Крошилина С.В.

доц. Тишкина В.В.

**Цель работы**: разработать генератор гаммы на базе линейного рекуррентного регистра сдвига и выполнить шифрование путем наложения гаммы на открытый текст.

**Ход работы**

На рисунках 1 – 3 показаны схемы алгоритмов данной программы.

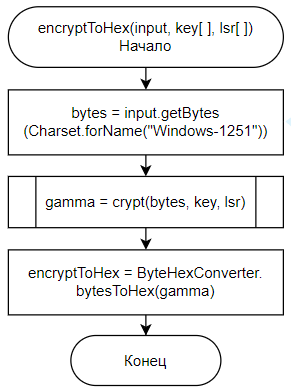


Рисунок 1 – Схема подпрограммы encryptToHex

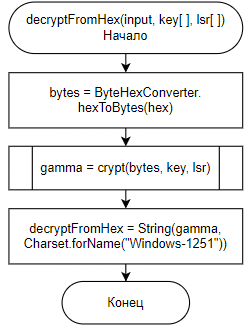


Рисунок 2 – Схема подпрограммы decryptFromHex

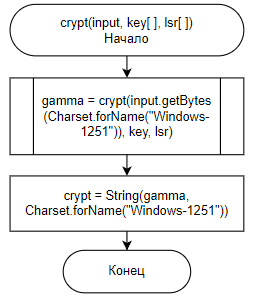


Рисунок 3 – Схема подпрограммы crypt

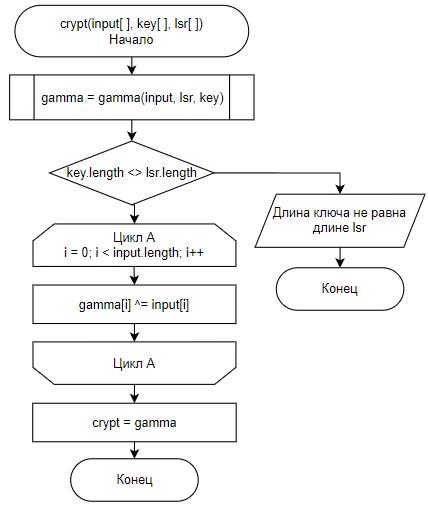


Рисунок 4 – Схема подпрограммы crypt

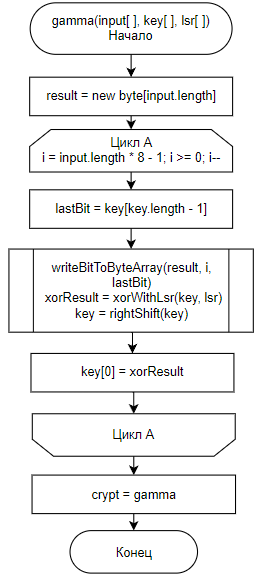


Рисунок 5 – Схема подпрограммы gamma

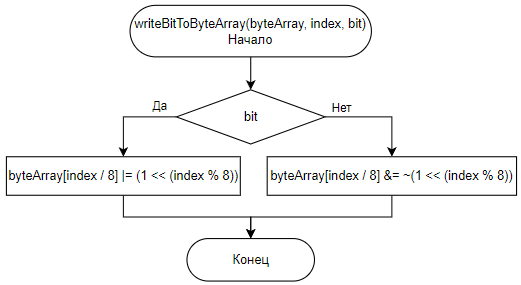


Рисунок 6 – Схема подпрограммы writeBitToByteArray

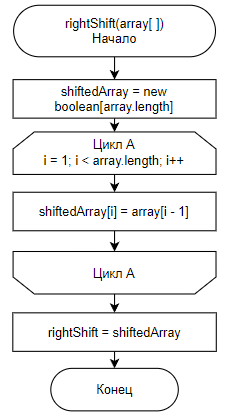


Рисунок 7 – Схема подпрограммы rightShift

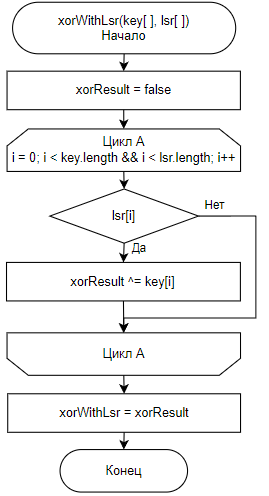


Рисунок 8 – Схема подпрограммы xorWithLsr

**Код программы:**

public class Gamma {  
 public static String encryptToHex(String input, boolean[] key, boolean[] lsr) {  
 var bytes = input.getBytes(Charset.*forName*("Windows-1251"));  
 byte[] gamma = *crypt*(bytes, key, lsr);  
 return ByteHexConverter.*bytesToHex*(gamma);  
 }  
  
 public static String decryptFromHex(String hex, boolean[] key, boolean[] lsr) {  
 var bytes = ByteHexConverter.*hexToBytes*(hex);  
 byte[] gamma = *crypt*(bytes, key, lsr);  
 return new String(gamma, Charset.*forName*("Windows-1251"));  
 }  
  
 public static String crypt(String input, boolean[] key, boolean[] lsr) {  
 byte[] gamma = *crypt*(input.getBytes(Charset.*forName*("Windows-1251")), key, lsr);  
 return new String(gamma, Charset.*forName*("Windows-1251"));  
 }  
  
 private static byte[] crypt(byte[] input, boolean[] key, boolean[] lsr) {  
 if (key.length != lsr.length) throw new IllegalArgumentException("Длина ключа не равна длине lsr");  
 byte[] gamma = *gamma*(input, lsr, key);  
  
 for (int i = 0; i < input.length; i++) {  
 gamma[i] ^= input[i];  
 }  
 return gamma;  
 }  
  
 private static byte[] gamma(byte[] input, boolean[] lsr, boolean[] key) {  
 byte[] result = new byte[input.length];  
 for (int i = input.length \* 8 - 1; i >= 0; i--) {  
 boolean lastBit = key[key.length - 1];  
 *writeBitToByteArray*(result, i, lastBit);  
 boolean xorResult = *xorWithLsr*(key, lsr);  
 key = *rightShift*(key);  
 key[0] = xorResult;  
 }  
  
 return result;  
 }  
  
 private static boolean[] rightShift(boolean[] array) {  
 boolean[] shiftedArray = new boolean[array.length];  
  
 for (int i = 1; i < array.length; i++) {  
 shiftedArray[i] = array[i - 1]; // Копируем значение из предыдущего элемента  
 }  
  
 return shiftedArray;  
 }  
  
 private static void writeBitToByteArray(byte[] byteArray, int index, boolean bit) {  
 if (bit) {  
 byteArray[index / 8] |= (1 << (index % 8));  
 } else {  
 byteArray[index / 8] &= ~(1 << (index % 8));  
 }  
 }  
  
 private static boolean xorWithLsr(boolean[] key, boolean[] lsr) {  
 boolean xorResult = false;  
 for (int i = 0; i < key.length && i < lsr.length; i++) {  
 if (lsr[i]) { // Если значение в lsr равно true  
 xorResult ^= key[i]; // Выполняем операцию XOR с соответствующим элементом в key  
 }  
 }  
 return xorResult;  
 }  
}

**Вывод**: в ходе выполнения работы был разработан генератор гаммы на базе линейного рекуррентного регистра сдвига и выполнено шифрование путем наложения гаммы на открытый текст.