**Молдавский государственный университет**

**Факультет математики и информатики**

**Департамент Информатики**

**УПРАЖНЕНИЕ**

**RC2**

**Выполнил**: Гуцу Даниил.

**Группа:** IA2102

**Проверила:** др.,Чербу О.

Кишинев, 2023

**КИШИНЕВ – 2023**

RC2 (Rivest Cipher 2) - это симметричный блочный шифр, разработанный Роном Ривестом в 1987 году. Этот шифр был разработан для использования в качестве алгоритма симметричного шифрования, который может использоваться для защиты конфиденциальности данных, обеспечивая их конфиденциальность при передаче или хранении.

Вот основные характеристики RC2:

1. \*\*Симметричный шифр\*\*: RC2 использует один и тот же ключ для шифрования и дешифрования данных. Таким образом, отправитель и получатель должны иметь доступ к одному и тому же секретному ключу.

2. \*\*Блочный шифр\*\*: RC2 оперирует блоками данных фиксированного размера. Размер блока зависит от реализации RC2, но он обычно составляет 64 бита (8 байтов).

3. \*\*Переменная длина ключа\*\*: RC2 позволяет использовать ключи разной длины, от 1 до 128 бит. Большая длина ключа обычно означает более надежное шифрование.

4. \*\*Сложный алгоритм подстановки и перестановки\*\*: RC2 использует сложные процедуры подстановки и перестановки данных, что делает его устойчивым к различным атакам.

5. \*\*Раунды и ключи раунда\*\*: Шифр RC2 выполняет несколько раундов (обычно 16) с использованием разных ключей раунда, получаемых из исходного ключа с помощью процедуры расширения ключа.

RC2 был широко использован в прошлом, но с течением времени он был заменен более современными и надежными алгоритмами шифрования, такими как AES (Advanced Encryption Standard). Это связано с тем, что RC2 имел некоторые уязвимости и проблемы с безопасностью, и его использование не рекомендуется в современных приложениях и системах.

Код программы:

package rc2;  
  
import java.math.BigInteger;  
import java.nio.charset.StandardCharsets;  
import java.util.ArrayList;  
import java.util.List;  
  
public class RC2 {  
 public static void main(String[] args) {  
 String text = "olegtipa";  
 String key = "ot";  
 System.*out*.println("Given text: " + text);  
 System.*out*.println("Given key: " + key);  
  
 String binaryText = new BigInteger(text.getBytes(StandardCharsets.*UTF\_8*)).toString(2);  
 binaryText = String.*format*("%64s", binaryText).replaceAll(" ", "0");  
 String binaryKey = new BigInteger(key.getBytes(StandardCharsets.*UTF\_8*)).toString(2);  
 binaryKey = String.*format*("%16s", binaryKey).replaceAll(" ", "0");  
 System.*out*.println("Given word in binary: " + *prettyBinary*(binaryText, 8, " "));  
 System.*out*.println("Given key in binary: " + *prettyBinary*(binaryKey, 8, " "));  
  
 StringBuilder a1 = new StringBuilder();  
 for (int i = 0; i < 16; ++i) {  
 a1.append(binaryText.charAt(i));  
 }  
 String a = a1.toString();  
  
 StringBuilder b1 = new StringBuilder();  
 for (int i = 16; i < 32; ++i) {  
 b1.append(binaryText.charAt(i));  
 }  
 String b = b1.toString();  
  
 StringBuilder c1 = new StringBuilder();  
 for (int i = 32; i < 48; ++i) {  
 c1.append(binaryText.charAt(i));  
 }  
 String c = c1.toString();  
  
 StringBuilder d1 = new StringBuilder();  
 for (int i = 48; i < 64; ++i) {  
 d1.append(binaryText.charAt(i));  
 }  
 String d = d1.toString();  
  
 System.*out*.println("a = " + *prettyBinary*(a, 8, " "));  
 System.*out*.println("b = " + *prettyBinary*(b, 8, " "));  
 System.*out*.println("c = " + *prettyBinary*(c, 8, " "));  
 System.*out*.println("d = " + *prettyBinary*(d, 8, " "));  
  
 // Step 1  
 int part1Int = (Integer.*parseInt*(a, 2) + Integer.*parseInt*(binaryKey, 2));  
 String part1 = Integer.*toBinaryString*(part1Int);  
 part1 = String.*format*("%32s", part1).replaceAll(" ", "0");  
 System.*out*.println("\nStep 1: a + key = " + part1Int + " = " + part1);  
  
 // Step 2  
 int part2Int = Integer.*parseInt*(c, 2) & Integer.*parseInt*(d, 2);  
 String part2 = Integer.*toBinaryString*(part2Int);  
 part2 = String.*format*("%32s", part2).replaceAll(" ", "0");  
 System.*out*.println("\nStep 2: c ^ d = " + part2);  
  
 //Step 3  
 int part3Int = ~Integer.*parseInt*(d, 2) & Integer.*parseInt*(b, 2);  
 String part3 = Integer.*toBinaryString*(part3Int);  
 part3 = String.*format*("%32s", part3).replaceAll(" ", "0");  
 System.*out*.println("\nStep 3: (not d) ^ b = " + part3);  
  
 //Step 4  
 int part4Int = part2Int + part3Int;  
 String part4 = Integer.*toBinaryString*(part4Int);  
 part4 = String.*format*("%32s", part4).replaceAll(" ", "0");  
 System.*out*.println("\nStep 4: (c ^ d) + ((not d) ^ b) = " + part4);  
  
 // Step 5  
 int part5Int = part1Int + part4Int;  
 String part5 = Integer.*toBinaryString*(part5Int);  
 part5 = String.*format*("%32s", part5).replaceAll(" ", "0");  
 System.*out*.println("\nStep 5: a + key + (c ^ d) + ((not d) ^ b) = " + part5);  
  
 // Step 6  
 int part6Int = part5Int << 4;  
 String part6 = Integer.*toBinaryString*(part6Int);  
 part6 = String.*format*("%32s", part6).replaceAll(" ", "0");  
 System.*out*.println("\nStep 6: part5 <<< 4 = " + part6);  
  
 // Step 7  
 a = part6;  
 System.*out*.println("b => " + b + "\n" + "c => " + c + "\n" + "d => " + d + "\n" + "a => " + a);  
  
 // Step 8  
 String bcd = b.concat(c).concat(d);  
 String encryption = bcd.concat(a);  
 System.*out*.println("Concatenating b, c, d, a => " + *prettyBinary*(encryption, 8, " "));  
  
 // Result  
 StringBuilder result = new StringBuilder();  
 int aDecimal = Integer.*parseInt*(a, 2);  
 String aHex = Integer.*toString*(aDecimal, 16);  
 for (int i = 0; i < bcd.length()/8; i++) {  
 int az = Integer.*parseInt*(bcd.substring(8\*i,(i+1)\*8),2);  
 result.append((char) (az));  
 }  
 System.*out*.println("Result = " + "\"" + result + "\"" + " + \"" + aHex + "\"");  
  
 // Decryption  
 System.*out*.println("\n===================DECRYPTION===================");  
  
 //Step 1  
 part5Int = part6Int >>> 4;  
 part5 = Integer.*toBinaryString*(part5Int);  
 part5 = String.*format*("%32s", part5).replaceAll(" ", "0");  
 System.*out*.println("Step 1: part6 >>> 4 = " + part5);  
  
 // Step 2  
 System.*out*.println("\nStep 2: c ^ d = " + part2);  
  
 //Step 3  
 System.*out*.println("\nStep 3: (not d) ^ b = " + part3);  
  
 //Step 4  
 System.*out*.println("\nStep 4: (c ^ d) + ((not d) ^ b) = " + part4);  
  
 // Step 5  
 int part5IntDecrypt = part5Int - part4Int;  
 String part5Decrypt = Integer.*toBinaryString*(part5IntDecrypt);  
 part5Decrypt = String.*format*("%32s", part5Decrypt).replaceAll(" ", "0");  
 System.*out*.println("\nStep 5: (a - (b ^ (not d) + (c ^ d) )) => " + part5IntDecrypt + " = " + part5Decrypt);  
  
 // Step 6  
 part6Int = part5IntDecrypt - Integer.*parseInt*(binaryKey, 2);  
 part6 = Integer.*toBinaryString*(part6Int);  
 part6 = String.*format*("%16s", part6).replaceAll(" ", "0");  
 System.*out*.println("\nStep 6: part5 - CV => " + part6Int + " = " + part6);  
  
 // Step 7  
 result = new StringBuilder();  
 String abcd = part6.concat(b).concat(c).concat(d);  
 for (int i = 0; i < abcd.length()/8; i++) {  
 int az = Integer.*parseInt*(abcd.substring(8\*i,(i+1)\*8),2);  
 result.append((char) (az));  
 }  
 System.*out*.println("Result = " + "\"" + result + "\"");  
 }  
  
 public static String prettyBinary(String binary, int blockSize, String separator) {  
 List<String> result = new ArrayList<>();  
 int index = 0;  
 while (index < binary.length()) {  
 result.add(binary.substring(index, Math.*min*(index + blockSize, binary.length())));  
 index += blockSize;  
 }  
 return String.*join*(separator, result);  
 }  
}

Вывод праграммы:

Given text: criptogg

Given key: GD

Given word in binary: 01100011 01110010 01101001 01110000 01110100 01101111 01100111 01100111

Given key in binary: 01000111 01000100

a = 01100011 01110010

b = 01101001 01110000

c = 01110100 01101111

d = 01100111 01100111

Step 1: a + key = 43702 = 00000000000000001010101010110110

Step 2: c ^ d = 00000000000000000110010001100111

Step 3: (not d) ^ b = 00000000000000000000100000010000

Step 4: (c ^ d) + ((not d) ^ b) = 00000000000000000110110001110111

Step 5: a + key + (c ^ d) + ((not d) ^ b) = 00000000000000010001011100101101

Step 6: part5 <<< 4 = 00000000000100010111001011010000

b => 0110100101110000

c => 0111010001101111

d => 0110011101100111

a => 00000000000100010111001011010000

Concatenating b, c, d, a => 01101001 01110000 01110100 01101111 01100111 01100111 00000000 00010001 01110010 11010000

Result = "iptogg" + "1172d0"

===================DECRYPTION===================

Step 1: part6 >>> 4 = 00000000000000010001011100101101

Step 2: c ^ d = 00000000000000000110010001100111

Step 3: (not d) ^ b = 00000000000000000000100000010000

Step 4: (c ^ d) + ((not d) ^ b) = 00000000000000000110110001110111

Step 5: (a - (b ^ (not d) + (c ^ d) )) => 43702 = 00000000000000001010101010110110

Step 6: part5 - CV => 25458 = 0110001101110010

Result = "criptogg"