**Молдавский государственный университет**

**Факультет математики и информатики**

**Департамент Информатики**

**УПРАЖНЕНИЕ**

**RC4**

**Выполнил**: Гуцу Даниил.

**Группа:** IA2102

**Проверила:** др.,Чербу О.

Кишинев, 2023

**КИШИНЕВ – 2023**

RC4 (Rivest Cipher 4) - это симметричный потоковый шифр, разработанный Роном Ривестом в 1987 году. RC4 был широко использован в прошлом для шифрования данных, включая безопасную передачу данных по Интернету, но с течением времени он стал менее рекомендуемым в силу выявленных уязвимостей и атак на этот шифр.

Вот основные характеристики RC4:

1. \*\*Симметричный потоковый шифр\*\*: RC4 генерирует псевдослучайный ключевой поток, который затем комбинируется с открытым текстом (или другими данными) побитово. Полученное шифрованное сообщение является результатом операции XOR между ключевым потоком и открытым текстом.

2. \*\*Переменная длина ключа\*\*: RC4 позволяет использовать ключи разной длины, обычно от 40 до 256 бит. Большая длина ключа увеличивает уровень безопасности.

3. \*\*Простой алгоритм\*\*: RC4 известен своей простотой и относительно высокой скоростью работы. Это делает его привлекательным для быстрого шифрования данных.

4. \*\*Несколько раундов\*\*: В отличие от некоторых других симметричных шифров, RC4 не имеет фиксированного количества раундов. Вместо этого он просто генерирует ключевой поток и применяет его для шифрования данных.

Несмотря на свою простоту и скорость, RC4 имеет некоторые серьезные уязвимости и проблемы с безопасностью, которые были выявлены в ходе исследований. Эти уязвимости включают в себя проблемы в начальной инициализации ключа и корреляцию ключевого потока, что делает его небезопасным для многих криптографических приложений. Из-за этого рекомендуется использовать более современные алгоритмы шифрования, такие как AES (Advanced Encryption Standard), вместо RC4 в современных системах и приложениях.

Код программы:

package rc4;  
  
import java.math.BigInteger;  
import java.util.Scanner;  
  
public class RC4 {  
 static int[] *S* = {1, 2, 3, 4, 5, 6, 7, 8};  
 public static void main(String args[])  
 {  
 *S* = new int[]{1, 2, 3, 4, 5, 6, 7, 8};  
 Scanner s = new Scanner(System.*in*);  
 String key = *inputKey*(s);  
 String message = *inputMessage*(s);  
 s.close();  
  
 int[] keyASCII = *charToASCII*(key.toCharArray());  
  
 *showASCIIKey*(keyASCII);  
 *showFullInitalData*(keyASCII, *S*);  
  
 *S* = *swapping*(*S*, keyASCII);  
  
 int t = *tWord*(*S*);  
  
 *showBinarKey*(keyASCII[t], t, keyASCII);  
  
 String[] r = *encrypt*(message.split(""), keyASCII[t]);  
  
 *ShowEncryptingOfNORMandK*(r, message.split(""), keyASCII[t], t);  
  
 String[] m = *descript*(r, keyASCII[t]);  
  
 *ShowDescriptingOfNORMandK*(r, m, keyASCII[t], t);  
  
 *showTheResultOfDescripting*(m, key);  
 }  
  
 private static String inputKey(Scanner s)  
 {  
 String key;  
  
 do {  
 System.*out*.print("Input the key (8 numbers, without space): ");  
 key = s.nextLine();  
 }while(key.length()!=8);  
  
 return key;  
 }  
  
 private static String inputMessage(Scanner s)  
 {  
 String message;  
  
 do {  
 System.*out*.print("Input the message (8 symbols, without space): ");  
 message = s.nextLine();  
 }while(message.length()!=8);  
  
 return message;  
 }  
  
 private static int[] charToASCII(char... args)  
 {  
 int i = 0;  
 int ASCII[] = new int[args.length];  
  
 for(char arg : args)  
 {  
 ASCII[i++] = (int) arg;  
 }  
 return ASCII;  
 }  
  
 private static void showASCIIKey(int[] keyASCII)  
 {  
 System.*out*.println("\nThe key in ASCII: ");  
  
 for(int i: keyASCII)  
 {  
 System.*out*.print(i + "\t");  
 }  
  
 System.*out*.print("\n\n");  
 }  
  
 private static void showFullInitalData(int[] keyASCII, int[] S)  
 {  
 System.*out*.print("P: \t");  
 for(int i=0;i<8;i++)  
 {  
 System.*out*.print(i + "\t");  
 }  
  
 System.*out*.print("\nS: \t");  
 for(int i: S)  
 {  
 System.*out*.print(i + "\t");  
 }  
  
 System.*out*.print("\nK: \t");  
 for(int i: keyASCII)  
 {  
 System.*out*.print(i + "\t");  
 }  
  
 System.*out*.print("\n\n");  
 }  
  
 private static int[] swapping(int[] S, int[] keyASCII)  
 {  
 for(int i=0, j = 0; i<8 ; i++)  
 {  
 *showSwapping*(i, j, S, keyASCII);  
  
 j = (j + S[j] + keyASCII[i]) % 8;  
 *swap*(S, i, j);  
 swap(keyASCII, i, j);  
  
 showFullInitalData(keyASCII, S);  
 }  
  
 return S;  
 }  
  
 private static void showSwapping(int i, int j, int[] S, int[] keyASCII)  
 {  
 int newJ = (j + S[j] + keyASCII[i]) % 8;  
  
 System.out.println("\n" + (i+1) + ". j = (j + S[j] + K[i]) mod(%) 8 => " + j + " + " + S[j] + " + "  
 + keyASCII[i] + " mod(%) 8" + " = "  
 + (j + S[j] + keyASCII[i]) % 8);  
 System.out.println(" S - Swap( " + S[i] +", " + S[newJ] + " )");  
 System.out.println(" K - Swap( " + keyASCII[i] +", " + keyASCII[newJ] + " )");  
 }  
  
 private static void swap(int[] T, int i, int j)  
 {  
 int tmp = T[i];  
 T[i] = T[j];  
 T[j] = tmp;  
 }  
  
 private static int tWord(int[] S)  
 {  
 int i = 0, j = 0;  
 i = (i+1) % 8;  
 j = (j+S[i]) % 8;  
 int t = (S[i]+S[j]) % 8;  
 return t;  
 }  
  
 private static void showPseudoGeneration(int a, int b, int S[], int i, int j)  
 {  
 System.out.println("Generation of pseudo range: ");  
 System.out.println("i = (i+1) mod(%) 8 => " + "( " + a + " + 1 ) mod(%) 8 = " +(a+1) % 8);  
 System.out.println("j = (j+S[i]) mod(%) 8 => " + "( " + b + " + " + S[i] + " ) mod(%) 8 = " +(b+S[i]) % 8);  
 System.out.println("t = (S[i]+S[j]) % 8 => " + "( " + S[i] + " + " + S[j] + " ) mod(%) 8 = " +(S[i]+S[j]) % 8);  
 System.out.print("\n\n");  
 }  
  
 private static void showBinarKey(int binarTKey, int t, int[] keyASCII)  
 {  
 System.out.println("K[t] = K[" + t +"] = ("+ keyASCII[t] + ") = " + IntToBinarString(binarTKey) + "\n\n");  
 }  
  
 private static String[] encrypt(String[] message, int binarTKey)  
 {  
 String[] r = new String[8];  
 String sctbs;  
 int ictbs;  
  
 for(int i=0;i<8; i++)  
 {  
 sctbs = CharToBinarString(message[i]);  
 ictbs = BinarStringToInt(sctbs);  
 r[i] = BinaryStringToSChar(IntToBinarString (ictbs ^ binarTKey));  
 }  
  
 System.out.println("\n");  
 return r;  
 }  
  
 private static void showEncryptingOfMessage(char messageChar, int i)  
 {  
 int m = (int) messageChar;  
 System.out.println(i + ". m[" + (i-1) + "] = " + messageChar + " -> " + ((int) messageChar)+ " = " +  
 IntToBinarString(m));  
 }  
  
 private static void ShowEncryptingOfNORMandK(String[] r, String[] message, int binarTkey, int t)  
 {  
 for(int i=0;i<r.length;i++)  
 {  
 System.out.println(r[i]);  
 int m = (int) message[i].charAt(0);  
 String itbs = IntToBinarString (m ^ binarTkey);  
  
 System.out.println((i+1) + ". r["+i+"] = m["+i+"](="+message[i]+") XOR(^) K["+t+"] =>"  
 + IntToBinarString(m) + " XOR(^) " + IntToBinarString(binarTkey) +" = "  
 + itbs + " => " + BinarStringToInt(itbs) + " => " + BinaryStringToSChar(itbs));  
 }  
  
 System.out.println("\n");  
 }  
  
 private static String[] descript(String[] messageR, int binarTKey)  
 {  
 String[] m = new String[8];  
 String sctbs;  
 int ictbs;  
  
 for(int i=0;i<8; i++)  
 {  
 showDesciptingOfMessage(messageR[i].charAt(0), i+1);  
 sctbs = CharToBinarString(messageR[i]);  
 ictbs = BinarStringToInt(sctbs);  
 System.out.println(sctbs);  
 //m[i] = BinaryStringToSChar(IntToBinarString (ictbs ^ binarTKey));  
 }  
  
 System.out.println("\n");  
 return m;  
 }  
  
 private static void ShowDescriptingOfNORMandK(String[] r, String[] message, int binarTkey, int t)  
 {  
 String binarKey = IntToBinarString(binarTkey);  
  
 for(int i=0;i<message.length;i++)  
 {  
 int rMessage = (int) r[i].charAt(0);  
 String itbs = IntToBinarString (rMessage ^ binarTkey);  
  
 System.out.println((i+1) + ". m["+i+"] = r["+i+"](="+r[i]+") XOR(^) K["+t+"] =>"  
 + IntToBinarString(rMessage) + " XOR(^) " + binarKey +" = "  
 + itbs + " => " + BinarStringToInt(itbs) + " => " + BinaryStringToSChar(itbs));  
 }  
  
 System.out.println("\n");  
  
 }  
  
 private static void showDesciptingOfMessage(char messageChar, int i)  
 {  
 int r = (int) messageChar;  
 System.out.println(i + ". r[" + (i-1) + "] = " + messageChar + " -> " + ((int) messageChar)+ " = " +  
 IntToBinarString(r));  
 }  
  
 private static String CharToBinarString(String first) {  
 String binarString = new BigInteger(first.getBytes()).toString(2);  
 return binarString;  
 }  
  
 private static int BinarStringToInt(String first) {  
 return Integer.parseInt(first, 2);  
 }  
  
 private static String IntToBinarString(int first) {  
 String binarString = Integer.toBinaryString(first);  
 return binarString;  
 }  
  
 private static String BinaryStringToSChar(String first) {  
 String text = new String(new BigInteger(first, 2).toByteArray());  
 return text;  
 }  
  
 private static void showTheResultOfDescripting(String[] m, String key)  
 {  
 System.out.println("Your encrypted word is: "+concatArrayString(m));  
 System.out.println("The key is: " + key);  
 }  
  
 private static String concatArrayString(String[] m)  
 {  
 String str = "";  
 for(String i : m)  
 {  
 str += i;  
 }  
  
 return str;  
 }  
}

Вывод программы:

The key in ASCII:

51 53 54 56 51 53 54 56

P: 0 1 2 3 4 5 6 7

S: 1 2 3 4 5 6 7 8

K: 51 53 54 56 51 53 54 56

1. j = (j + S[j] + K[i]) mod(%) 8 => 0 + 1 + 51 mod(%) 8 = 4

S - Swap( 1, 5 )

K - Swap( 51, 51 )

P: 0 1 2 3 4 5 6 7

S: 5 2 3 4 1 6 7 8

K: 51 53 54 56 51 53 54 56

2. j = (j + S[j] + K[i]) mod(%) 8 => 4 + 1 + 53 mod(%) 8 = 2

S - Swap( 2, 3 )

K - Swap( 53, 54 )

P: 0 1 2 3 4 5 6 7

S: 5 3 2 4 1 6 7 8

K: 51 54 53 56 51 53 54 56

3. j = (j + S[j] + K[i]) mod(%) 8 => 2 + 2 + 53 mod(%) 8 = 1

S - Swap( 2, 3 )

K - Swap( 53, 54 )

P: 0 1 2 3 4 5 6 7

S: 5 2 3 4 1 6 7 8

K: 51 53 54 56 51 53 54 56

4. j = (j + S[j] + K[i]) mod(%) 8 => 1 + 2 + 56 mod(%) 8 = 3

S - Swap( 4, 4 )

K - Swap( 56, 56 )

P: 0 1 2 3 4 5 6 7

S: 5 2 3 4 1 6 7 8

K: 51 53 54 56 51 53 54 56

5. j = (j + S[j] + K[i]) mod(%) 8 => 3 + 4 + 51 mod(%) 8 = 2

S - Swap( 1, 3 )

K - Swap( 51, 54 )

P: 0 1 2 3 4 5 6 7

S: 5 2 1 4 3 6 7 8

K: 51 53 51 56 54 53 54 56

6. j = (j + S[j] + K[i]) mod(%) 8 => 2 + 1 + 53 mod(%) 8 = 0

S - Swap( 6, 5 )

K - Swap( 53, 51 )

P: 0 1 2 3 4 5 6 7

S: 6 2 1 4 3 5 7 8

K: 53 53 51 56 54 51 54 56

7. j = (j + S[j] + K[i]) mod(%) 8 => 0 + 6 + 54 mod(%) 8 = 4

S - Swap( 7, 3 )

K - Swap( 54, 54 )

P: 0 1 2 3 4 5 6 7

S: 6 2 1 4 7 5 3 8

K: 53 53 51 56 54 51 54 56

8. j = (j + S[j] + K[i]) mod(%) 8 => 4 + 7 + 56 mod(%) 8 = 3

S - Swap( 8, 4 )

K - Swap( 56, 56 )

P: 0 1 2 3 4 5 6 7

S: 6 2 1 8 7 5 3 4

K: 53 53 51 56 54 51 54 56

K[t] = K[3] = (56) = 111000

[

1. r[0] = m[0](=c) XOR(^) K[3] =>1100011 XOR(^) 111000 = 1011011 => 91 => [

J

2. r[1] = m[1](=r) XOR(^) K[3] =>1110010 XOR(^) 111000 = 1001010 => 74 => J

Q

3. r[2] = m[2](=i) XOR(^) K[3] =>1101001 XOR(^) 111000 = 1010001 => 81 => Q

H

4. r[3] = m[3](=p) XOR(^) K[3] =>1110000 XOR(^) 111000 = 1001000 => 72 => H

L

5. r[4] = m[4](=t) XOR(^) K[3] =>1110100 XOR(^) 111000 = 1001100 => 76 => L

W

6. r[5] = m[5](=o) XOR(^) K[3] =>1101111 XOR(^) 111000 = 1010111 => 87 => W

\_

7. r[6] = m[6](=g) XOR(^) K[3] =>1100111 XOR(^) 111000 = 1011111 => 95 => \_

\_

8. r[7] = m[7](=g) XOR(^) K[3] =>1100111 XOR(^) 111000 = 1011111 => 95 => \_

1. r[0] = [ -> 91 = 1011011

1011011

2. r[1] = J -> 74 = 1001010

1001010

3. r[2] = Q -> 81 = 1010001

1010001

4. r[3] = H -> 72 = 1001000

1001000

5. r[4] = L -> 76 = 1001100

1001100

6. r[5] = W -> 87 = 1010111

1010111

7. r[6] = \_ -> 95 = 1011111

1011111

8. r[7] = \_ -> 95 = 1011111

1011111

1. m[0] = r[0](=[) XOR(^) K[3] =>1011011 XOR(^) 111000 = 1100011 => 99 => c

2. m[1] = r[1](=J) XOR(^) K[3] =>1001010 XOR(^) 111000 = 1110010 => 114 => r

3. m[2] = r[2](=Q) XOR(^) K[3] =>1010001 XOR(^) 111000 = 1101001 => 105 => i

4. m[3] = r[3](=H) XOR(^) K[3] =>1001000 XOR(^) 111000 = 1110000 => 112 => p

5. m[4] = r[4](=L) XOR(^) K[3] =>1001100 XOR(^) 111000 = 1110100 => 116 => t

6. m[5] = r[5](=W) XOR(^) K[3] =>1010111 XOR(^) 111000 = 1101111 => 111 => o

7. m[6] = r[6](=\_) XOR(^) K[3] =>1011111 XOR(^) 111000 = 1100111 => 103 => g

8. m[7] = r[7](=\_) XOR(^) K[3] =>1011111 XOR(^) 111000 = 1100111 => 103 => g