**Молдавский государственный университет**

**Факультет математики и информатики**

**Департамент Информатики**

**УПРАЖНЕНИЕ**

**Tea**

**Выполнил**: Гуцу Даниил.

**Группа:** IA2102

**Проверила:** др.,Чербу О.

Кишинев, 2023

**КИШИНЕВ – 2023**

TEA (Tiny Encryption Algorithm) — это симметричный блочный шифр, который был разработан Дэвидом Уиллшаем (David Wheeler) и Роджером Нидхемом (Roger Needham) в 1994 году. Этот шифр является одним из примеров шифрования с открытым исходным кодом и изначально разрабатывался с целью обеспечить легкий и быстрый способ шифрования данных на маломощных устройствах, таких как смарт-карты и встроенные системы.

Вот основные характеристики TEA:

1. \*\*Симметричный блочный шифр\*\*: TEA оперирует блоками данных фиксированного размера. Размер блока составляет 64 бита (8 байтов).

2. \*\*Использование ключа\*\*: TEA использует секретный ключ для шифрования и дешифрования данных. Ключ также имеет размер 128 бит (16 байт).

3. \*\*Простой алгоритм\*\*: TEA известен своей простотой и скоростью работы. Это делает его подходящим для маломощных устройств и систем с ограниченными вычислительными ресурсами.

4. \*\*Раунды и ключи раунда\*\*: TEA выполняет несколько раундов (обычно 64) с использованием ключей раунда, получаемых из исходного ключа. Каждый раунд включает в себя операции смешивания данных и ключей.

5. \*\*Ограниченная стойкость\*\*: TEA предоставляет базовую стойкость к атакам, но он не является криптографически надежным шифром для современных приложений без дополнительных мер защиты.

Несмотря на свою простоту и быстродействие, TEA не рекомендуется для использования в криптографических приложениях, требующих высокой стойкости к атакам, так как он был подвергнут различным криптоаналитическим атакам и уязвимостям. Вместо TEA для современных приложений рекомендуется использовать более надежные и стойкие алгоритмы шифрования, такие как AES (Advanced Encryption Standard).

Код программы:

package tea;  
  
import java.util.Arrays;  
import java.util.List;  
import java.util.stream.Collectors;  
  
public class Tea {  
 public static final int *DELTA* = 0x9e3779b9;  
 public static final long *MASK32* = (1L << 32) - 1;  
  
 public static void main(String[] args) {  
 System.*out*.println("δ = " + *DELTA*);  
  
 final String[] keyStrings = {"otgr", "lala", "trau", "tvar"};  
 System.*out*.println("Keys: " + Arrays.*toString*(keyStrings));  
  
 final String message = "olegtipa";  
 System.*out*.println("Message: " + message);  
  
 List<String> binaryMessageList = *toBinaryStringList*(message);  
 String binaryMessage = *toString*(binaryMessageList);  
 System.*out*.println("Message in binary: " + binaryMessage);  
  
 List<String> leftMessagePart = binaryMessageList.subList(0, message.length() / 2);  
 List<String> rightMessagePart = binaryMessageList.subList(message.length() / 2, message.length());  
 System.*out*.println("Left side: " + leftMessagePart);  
 System.*out*.println("Right side: " + rightMessagePart);  
  
 List<List<String>> binaryKeys = *toBinaryStringList*(keyStrings);  
 for (int i = 0; i < binaryKeys.size(); i++) {  
 System.*out*.println("K" + i + ": " + binaryKeys.get(i));  
 }  
  
 int[] leftInt = *getIntsFromString*(message.substring(0, message.length() / 2));  
 int[] rightInt = *getIntsFromString*(message.substring(message.length() / 2));  
  
 int[] keyInts = Arrays.*stream*(keyStrings)  
 .flatMapToInt(key -> Arrays.*stream*(*getIntsFromString*(key)))  
 .toArray();  
 System.*out*.println(Arrays.*stream*(keyInts).mapToObj(Tea::*toBinaryString*).toList());  
  
 int currentDelta = 0;  
  
// for (int j = 0; j < leftInt.length; j++) {  
// for (int i = 0; i < 32; ++i) {  
// currentDelta += DELTA;  
// leftInt[j] += ((rightInt[j] << 4) + keyInts[0]) ^ (rightInt[j] + currentDelta) ^ ((rightInt[j] >> 5) + keyInts[1]);  
// rightInt[j] += ((leftInt[j] << 4) + keyInts[2]) ^ (leftInt[j] + currentDelta) ^ ((leftInt[j] >> 5) + keyInts[3]);  
// }  
// }  
  
// currentDelta = encrypt(leftInt, rightInt, keyInts, currentDelta);  
  
 currentDelta += *DELTA*;  
 System.*out*.println(*toBinaryString*(rightInt[0] << 4));  
 System.*out*.println(*toBinaryString*((rightInt[0] << 4) + keyInts[0]));  
 System.*out*.println(*toBinaryString*(rightInt[0] + currentDelta));  
 System.*out*.println(*toBinaryString*(((rightInt[0] << 4) + keyInts[0]) ^ (rightInt[0] + currentDelta)));  
 System.*out*.println(*toBinaryString*(rightInt[0] >> 5));  
 System.*out*.println(*toBinaryString*((rightInt[0] >> 5) + keyInts[1]));  
 System.*out*.println(*toBinaryString*(((rightInt[0] << 4) + keyInts[0]) ^ (rightInt[0] + currentDelta) ^ ((rightInt[0] >> 5) + keyInts[1])));  
 System.*out*.println(*toBinaryString*(leftInt[0] + (((rightInt[0] << 4) + keyInts[0]) ^ (rightInt[0] + currentDelta) ^ ((rightInt[0] >> 5) + keyInts[1]))));  
  
 leftInt[0] += ((rightInt[0] << 4) + keyInts[0]) ^ (rightInt[0] + currentDelta) ^ ((rightInt[0] >> 5) + keyInts[1]);  
  
 System.*out*.println("1: right << 4\n" + *toBinaryString*(leftInt[0]) + " << 4");  
 System.*out*.println(*toBinaryString*(leftInt[0] << 4) + "\n");  
  
 System.*out*.println("2: ((1) + K2) mod 2^32\n" + *toBinaryString*(leftInt[0] << 4) + " + ");  
 System.*out*.println(*toBinaryString*(keyInts[2]) + " = ");  
 System.*out*.println(*toBinaryString*((leftInt[0] << 4) + keyInts[2]) + "\n");  
  
 System.*out*.println("3: (right + DELTA) mod 2^32\n" + *toBinaryString*(leftInt[0]) + " + ");  
 System.*out*.println(*toBinaryString*(*DELTA*) + " = ");  
 System.*out*.println(*toBinaryString*(leftInt[0] + *DELTA*) + "\n");  
  
 System.*out*.println("4: (2) xor (3)\n" + *toBinaryString*((leftInt[0] << 4) + keyInts[2]) + " xor ");  
 System.*out*.println(*toBinaryString*(leftInt[0] + *DELTA*) + " = ");  
 System.*out*.println(*toBinaryString*(((leftInt[0] << 4) + keyInts[2]) ^ (leftInt[0] + *DELTA*)) + "\n");  
  
 System.*out*.println("5: right >> 5\n" + *toBinaryString*(leftInt[0]) + " >> 5");  
 System.*out*.println(*toBinaryString*(leftInt[0] >>5) + "\n");  
  
 System.*out*.println("6: ((5) + K3) mod 2^32\n" + *toBinaryString*(leftInt[0] >> 5) + " + ");  
 System.*out*.println(*toBinaryString*(keyInts[3]) + " = ");  
 System.*out*.println(*toBinaryString*((leftInt[0] >> 5) + keyInts[3]) + "\n");  
  
 System.*out*.println("7: (4) xor (6)\n" + *toBinaryString*(((leftInt[0] << 4) + keyInts[2]) ^ (leftInt[0] + *DELTA*)) + " xor ");  
 System.*out*.println(*toBinaryString*((leftInt[0] >> 5) + keyInts[3]) + " = ");  
 System.*out*.println(*toBinaryString*(((leftInt[0] << 4) + keyInts[2]) ^ (leftInt[0] + *DELTA*) ^ ((leftInt[0] >> 5) + keyInts[3])) + "\n");  
  
 System.*out*.println("8: (left + (7)) mod 2^32\n" + *toBinaryString*(rightInt[0]) + " + ");  
 System.*out*.println(*toBinaryString*((((leftInt[0] << 4) + keyInts[2]) ^ (leftInt[0] + currentDelta) ^ ((leftInt[0] >> 5) + keyInts[3]))) + " = ");  
 System.*out*.println(*toBinaryString*(rightInt[0] + (((leftInt[0] << 4) + keyInts[2]) ^ (leftInt[0] + currentDelta) ^ ((leftInt[0] >> 5) + keyInts[3]))) + "\n");  
  
 rightInt[0] += ((leftInt[0] << 4) + keyInts[2]) ^ (leftInt[0] + currentDelta) ^ ((leftInt[0] >> 5) + keyInts[3]);  
  
 System.*out*.println(*toBinaryString*(leftInt[0]) + *toBinaryString*(rightInt[0]));  
  
 long[] result = new long[leftInt.length];  
 for (int i = 0; i < result.length; i++) {  
 result[i] = (leftInt[i] & *MASK32*) << 32 | (rightInt[i] & *MASK32*);  
 }  
  
 System.*out*.println("Result: ");  
 for (long l : result) {  
 System.*out*.print(*toBinaryString*(l) + ", ");  
 }  
  
 System.*out*.println();  
 for (long l : result) {  
 System.*out*.print(*longToString*(l));  
 }  
  
 System.*out*.println();  
  
 System.*out*.println("1: right << 4\n" + *toBinaryString*(leftInt[0]) + " << 4");  
 System.*out*.println(*toBinaryString*(leftInt[0] << 4) + "\n");  
  
 System.*out*.println("2: ((1) + K2) mod 2^32\n" + *toBinaryString*(leftInt[0] << 4) + " + ");  
 System.*out*.println(*toBinaryString*(keyInts[2]) + " = ");  
 System.*out*.println(*toBinaryString*((leftInt[0] << 4) + keyInts[2]) + "\n");  
  
 System.*out*.println("3: (right + DELTA) mod 2^32\n" + *toBinaryString*(leftInt[0]) + " + ");  
 System.*out*.println(*toBinaryString*(*DELTA*) + " = ");  
 System.*out*.println(*toBinaryString*(leftInt[0] + *DELTA*) + "\n");  
  
 System.*out*.println("4: (2) xor (3)\n" + *toBinaryString*((leftInt[0] << 4) + keyInts[2]) + " xor ");  
 System.*out*.println(*toBinaryString*(leftInt[0] + *DELTA*) + " = ");  
 System.*out*.println(*toBinaryString*(((leftInt[0] << 4) + keyInts[2]) ^ (leftInt[0] + *DELTA*)) + "\n");  
  
 System.*out*.println("5: right >> 5\n" + *toBinaryString*(leftInt[0]) + " >> 5");  
 System.*out*.println(*toBinaryString*(leftInt[0] >>5) + "\n");  
  
 System.*out*.println("6: ((5) + K3) mod 2^32\n" + *toBinaryString*(leftInt[0] >> 5) + " + ");  
 System.*out*.println(*toBinaryString*(keyInts[3]) + " = ");  
 System.*out*.println(*toBinaryString*((leftInt[0] >> 5) + keyInts[3]) + "\n");  
  
 System.*out*.println("7: (4) xor (6)\n" + *toBinaryString*(((leftInt[0] << 4) + keyInts[2]) ^ (leftInt[0] + *DELTA*)) + " xor ");  
 System.*out*.println(*toBinaryString*((leftInt[0] >> 5) + keyInts[3]) + " = ");  
 System.*out*.println(*toBinaryString*(((leftInt[0] << 4) + keyInts[2]) ^ (leftInt[0] + *DELTA*) ^ ((leftInt[0] >> 5) + keyInts[3])) + "\n");  
  
 System.*out*.println("8: (left - (7)) mod 2^32\n" + *toBinaryString*(rightInt[0]) + " - ");  
 System.*out*.println(*toBinaryString*((((leftInt[0] << 4) + keyInts[2]) ^ (leftInt[0] + currentDelta) ^ ((leftInt[0] >> 5) + keyInts[3]))) + " = ");  
 System.*out*.println(*toBinaryString*(rightInt[0] - (((leftInt[0] << 4) + keyInts[2]) ^ (leftInt[0] + currentDelta) ^ ((leftInt[0] >> 5) + keyInts[3]))) + "\n");  
  
 rightInt[0] -= ((leftInt[0] << 4) + keyInts[2]) ^ (leftInt[0] + currentDelta) ^ ((leftInt[0] >> 5) + keyInts[3]);  
  
 System.*out*.println("1: right << 4\n" + *toBinaryString*(leftInt[0]) + " << 4");  
 System.*out*.println(*toBinaryString*(leftInt[0] << 4) + "\n");  
  
 System.*out*.println("2: ((1) + K2) mod 2^32\n" + *toBinaryString*(leftInt[0] << 4) + " + ");  
 System.*out*.println(*toBinaryString*(keyInts[2]) + " = ");  
 System.*out*.println(*toBinaryString*((leftInt[0] << 4) + keyInts[2]) + "\n");  
  
 System.*out*.println("3: (right + DELTA) mod 2^32\n" + *toBinaryString*(leftInt[0]) + " + ");  
 System.*out*.println(*toBinaryString*(*DELTA*) + " = ");  
 System.*out*.println(*toBinaryString*(leftInt[0] + *DELTA*) + "\n");  
  
 System.*out*.println("4: (2) xor (3)\n" + *toBinaryString*((leftInt[0] << 4) + keyInts[2]) + " xor ");  
 System.*out*.println(*toBinaryString*(leftInt[0] + *DELTA*) + " = ");  
 System.*out*.println(*toBinaryString*(((leftInt[0] << 4) + keyInts[2]) ^ (leftInt[0] + *DELTA*)) + "\n");  
  
 System.*out*.println("5: right >> 5\n" + *toBinaryString*(leftInt[0]) + " >> 5");  
 System.*out*.println(*toBinaryString*(leftInt[0] >>5) + "\n");  
  
 System.*out*.println("6: ((5) + K3) mod 2^32\n" + *toBinaryString*(leftInt[0] >> 5) + " + ");  
 System.*out*.println(*toBinaryString*(keyInts[3]) + " = ");  
 System.*out*.println(*toBinaryString*((leftInt[0] >> 5) + keyInts[3]) + "\n");  
  
 System.*out*.println("7: (4) xor (6)\n" + *toBinaryString*(((leftInt[0] << 4) + keyInts[2]) ^ (leftInt[0] + *DELTA*)) + " xor ");  
 System.*out*.println(*toBinaryString*((leftInt[0] >> 5) + keyInts[3]) + " = ");  
 System.*out*.println(*toBinaryString*(((leftInt[0] << 4) + keyInts[2]) ^ (leftInt[0] + *DELTA*) ^ ((leftInt[0] >> 5) + keyInts[3])) + "\n");  
  
 System.*out*.println("8: (left - (7)) mod 2^32\n" + *toBinaryString*(rightInt[0]) + " - ");  
 System.*out*.println(*toBinaryString*((((leftInt[0] << 4) + keyInts[2]) ^ (leftInt[0] + currentDelta) ^ ((leftInt[0] >> 5) + keyInts[3]))) + " = ");  
 System.*out*.println(*toBinaryString*(rightInt[0] - (((leftInt[0] << 4) + keyInts[2]) ^ (leftInt[0] + currentDelta) ^ ((leftInt[0] >> 5) + keyInts[3]))) + "\n");  
  
 leftInt[0] -= ((rightInt[0] << 4) + keyInts[0]) ^ (rightInt[0] + currentDelta) ^ ((rightInt[0] >> 5) + keyInts[1]);  
 currentDelta -= *DELTA*;  
  
  
  
  
  
  
 System.*out*.println("Decryption:");  
// currentDelta = decrypt(leftInt, rightInt, keyInts, currentDelta);  
 for (int i : leftInt) {  
 System.*out*.print(*toBinaryString*(i));  
 }  
 for (int i : rightInt) {  
 System.*out*.print(*toBinaryString*(i));  
 }  
  
 StringBuilder stringBuilder = new StringBuilder();  
  
 for (int i : leftInt) {  
 stringBuilder.append(*intToString*(i));  
 }  
 for (int i : rightInt) {  
 stringBuilder.append(*intToString*(i));  
 }  
  
 System.*out*.println("\nMessage: " + stringBuilder);  
 }  
  
 public static List<List<String>> toBinaryStringList(String[] keys) {  
 return Arrays.*stream*(keys).map(Tea::*toBinaryStringList*).toList();  
 }  
  
 public static List<String> toBinaryStringList(String message) {  
 return message.chars()  
 .mapToObj(symbol -> String.*format*("%8s", Integer.*toBinaryString*(symbol))  
 .replace(" ", "0")  
 )  
 .toList();  
 }  
  
 public static String toBinaryString(String message) {  
 return message.chars()  
 .mapToObj(  
 symbol -> String.*format*("%8s", Integer.*toBinaryString*(symbol))  
 .replace(" ", "0")  
 )  
 .collect(Collectors.*joining*(" "));  
 }  
  
 public static String toBinaryString(long number) {  
 return String.*format*("%64s", Long.*toBinaryString*(number))  
 .replace(" ", "0")  
 .replaceAll("(.{8})", " $1");  
 }  
  
 public static String toBinaryString(int number) {  
 return String.*format*("%32s", Integer.*toBinaryString*(number))  
 .replace(" ", "0")  
 .replaceAll("(.{8})", " $1");  
 }  
  
 public static String longToString(long value) {  
 // create a char array of size 8  
 char[] chars = new char[8];  
  
 // loop through the array and assign each char with 16 bits of the long value  
 for (int i = 0; i < 8; i++) {  
 // use bit shifting and masking to get the 16 bits at the i-th position  
 chars[chars.length - i - 1] = (char) ((value >>> (i \* 8)) & 0xFFFF);  
 }  
  
 // return a new string from the char array  
 return new String(chars);  
 }  
  
 public static String intToString(int value) {  
 // create a char array of size 4  
 char[] chars = new char[4];  
  
 // loop through the array and assign each char with 8 bits of the long value  
 for (int i = 0; i < 4; i++) {  
 // use bit shifting and masking to get the 8 bits at the i-th position  
 chars[chars.length - i - 1] = (char) ((value >>> (i \* 8)) & 0xFF);  
 }  
  
 // return a new string from the char array  
 return new String(chars);  
 }  
  
 public static String toString(Iterable<String> strings) {  
 return String.*join*(" ", strings);  
 }  
  
 public static int[] getIntsFromString(String word) {  
 if (word == null || word.length() > 8 || word.isEmpty()) {  
 throw new IllegalArgumentException(word);  
 }  
  
 final int remainingSymbols = word.length() % 4;  
  
 final int arraySize = (remainingSymbols != 0) ?  
 word.length() / 4 + 1 :  
 word.length() / 4;  
  
 int[] result = new int[arraySize];  
 for (int j = 0, i = 0; j < arraySize && i < word.length() - 1; j++, i += 4) {  
 result[j] = (word.charAt(i) << 24) |  
 (word.charAt(i + 1) << 16) |  
 (word.charAt(i + 2) << 8) |  
 word.charAt(i + 3);  
 }  
  
 for (int i = 0; i < remainingSymbols; i++) {  
 result[arraySize - 1] = result[arraySize - 1] << 8 | word.charAt(word.length() - remainingSymbols);  
 }  
  
 return result;  
 }  
  
  
 public static int encrypt(int[] leftInt, int[] rightInt, int[] keyInts, int currentDelta) {  
 for (int j = 0; j < leftInt.length; j++) {  
 for (int i = 0; i < 32; ++i) {  
 currentDelta += *DELTA*;  
 leftInt[j] += ((rightInt[j] << 4) + keyInts[0]) ^ (rightInt[j] + currentDelta) ^ ((rightInt[j] >> 5) + keyInts[1]);  
 rightInt[j] += ((leftInt[j] << 4) + keyInts[2]) ^ (leftInt[j] + currentDelta) ^ ((leftInt[j] >> 5) + keyInts[3]);  
 }  
 }  
 return currentDelta;  
 }  
  
  
 public static int decrypt(int[] leftInt, int[] rightInt, int[] k, int currentDelta) {  
 int k0 = k[0], k1 = k[1], k2 = k[2], k3 = k[3]; /\* cache key \*/  
  
 for (int j = 0; j < leftInt.length; j++) {  
 for (int i = 0; i < 32; i++) { /\* basic cycle start \*/  
 rightInt[j] -= ((leftInt[j] << 4) + k2) ^ (leftInt[j] + currentDelta) ^ ((leftInt[j] >> 5) + k3);  
 leftInt[j] -= ((rightInt[j] << 4) + k0) ^ (rightInt[j] + currentDelta) ^ ((rightInt[j] >> 5) + k1);  
 currentDelta -= *DELTA*;  
 }  
 }  
  
 return currentDelta;  
 }  
}

Вывод программы:

δ = -1640531527

Keys: [gdgr, adua, teun, iver]

Message: criprogg

Message in binary: 01100011 01110010 01101001 01110000 01110010 01101111 01100111 01100111

Left side: [01100011, 01110010, 01101001, 01110000]

Right side: [01110010, 01101111, 01100111, 01100111]

K0: [01100111, 01100100, 01100111, 01110010]

K1: [01100001, 01100100, 01110101, 01100001]

K2: [01110100, 01100101, 01110101, 01101110]

K3: [01101001, 01110110, 01100101, 01110010]

[ 01100111 01100100 01100111 01110010, 01100001 01100100 01110101 01100001, 01110100 01100101 01110101 01101110, 01101001 01110110 01100101 01110010]

00100110 11110110 01110110 01110000

10001110 01011010 11011101 11100010

00010000 10100110 11100001 00100000

10011110 11111100 00111100 11000010

00000011 10010011 01111011 00111011

01100100 11110111 11110000 10011100

11111010 00001011 11001100 01011110

01011101 01111110 00110101 11001110

1: right << 4

01011101 01111110 00110101 11001110 << 4

11010111 11100011 01011100 11100000

2: ((1) + K2) mod 2^32

11010111 11100011 01011100 11100000 +

01110100 01100101 01110101 01101110 =

01001100 01001000 11010010 01001110

3: (right + DELTA) mod 2^32

01011101 01111110 00110101 11001110 +

10011110 00110111 01111001 10111001 =

11111011 10110101 10101111 10000111

4: (2) xor (3)

01001100 01001000 11010010 01001110 xor

11111011 10110101 10101111 10000111 =

10110111 11111101 01111101 11001001

5: right >> 5

01011101 01111110 00110101 11001110 >> 5

00000010 11101011 11110001 10101110

6: ((5) + K3) mod 2^32

00000010 11101011 11110001 10101110 +

01101001 01110110 01100101 01110010 =

01101100 01100010 01010111 00100000

7: (4) xor (6)

10110111 11111101 01111101 11001001 xor

01101100 01100010 01010111 00100000 =

11011011 10011111 00101010 11101001

8: (left + (7)) mod 2^32

01110010 01101111 01100111 01100111 +

11011011 10011111 00101010 11101001 =

01001110 00001110 10010010 01010000

01011101 01111110 00110101 11001110 01001110 00001110 10010010 01010000

Result:

01011101 01111110 00110101 11001110 01001110 00001110 10010010 01010000,

]嵾縵㗎칎与ຒ鉐

1: right << 4

01011101 01111110 00110101 11001110 << 4

11010111 11100011 01011100 11100000

2: ((1) + K2) mod 2^32

11010111 11100011 01011100 11100000 +

01110100 01100101 01110101 01101110 =

01001100 01001000 11010010 01001110

3: (right + DELTA) mod 2^32

01011101 01111110 00110101 11001110 +

10011110 00110111 01111001 10111001 =

11111011 10110101 10101111 10000111

4: (2) xor (3)

01001100 01001000 11010010 01001110 xor

11111011 10110101 10101111 10000111 =

10110111 11111101 01111101 11001001

5: right >> 5

01011101 01111110 00110101 11001110 >> 5

00000010 11101011 11110001 10101110

6: ((5) + K3) mod 2^32

00000010 11101011 11110001 10101110 +

01101001 01110110 01100101 01110010 =

01101100 01100010 01010111 00100000

7: (4) xor (6)

10110111 11111101 01111101 11001001 xor

01101100 01100010 01010111 00100000 =

11011011 10011111 00101010 11101001

8: (left - (7)) mod 2^32

01001110 00001110 10010010 01010000 -

11011011 10011111 00101010 11101001 =

01110010 01101111 01100111 01100111

1: right << 4

01011101 01111110 00110101 11001110 << 4

11010111 11100011 01011100 11100000

2: ((1) + K2) mod 2^32

11010111 11100011 01011100 11100000 +

01110100 01100101 01110101 01101110 =

01001100 01001000 11010010 01001110

3: (right + DELTA) mod 2^32

01011101 01111110 00110101 11001110 +

10011110 00110111 01111001 10111001 =

11111011 10110101 10101111 10000111

4: (2) xor (3)

01001100 01001000 11010010 01001110 xor

11111011 10110101 10101111 10000111 =

10110111 11111101 01111101 11001001

5: right >> 5

01011101 01111110 00110101 11001110 >> 5

00000010 11101011 11110001 10101110

6: ((5) + K3) mod 2^32

00000010 11101011 11110001 10101110 +

01101001 01110110 01100101 01110010 =

01101100 01100010 01010111 00100000

7: (4) xor (6)

10110111 11111101 01111101 11001001 xor

01101100 01100010 01010111 00100000 =

11011011 10011111 00101010 11101001

8: (left - (7)) mod 2^32

01110010 01101111 01100111 01100111 -

11011011 10011111 00101010 11101001 =

10010110 11010000 00111100 01111110

Decryption:

01100011 01110010 01101001 01110000 01110010 01101111 01100111 01100111

Message: criprogg