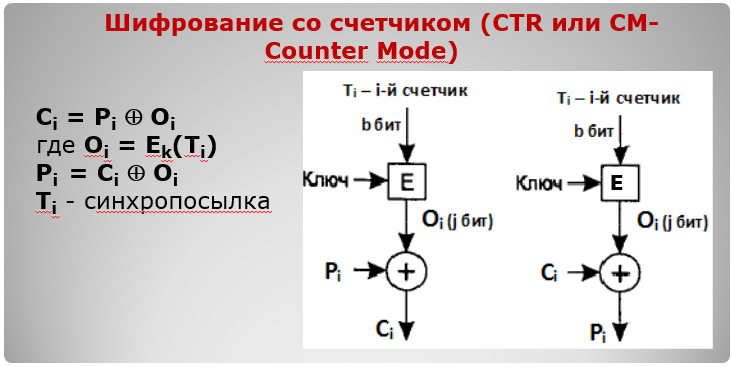
**Лабораторная работа № 4**

**РЕЖИМЫ РАБОТЫ БЛОЧНЫХ ШИФРОВ**

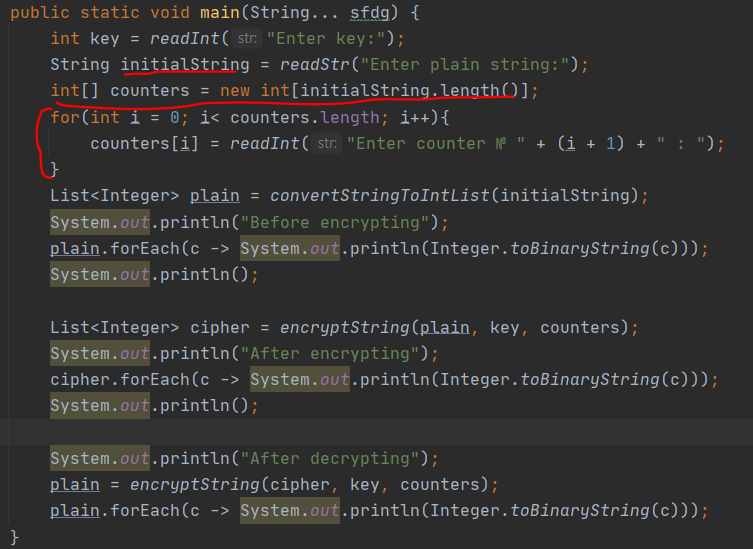
Цель работы. Изучить режимы работы блочных шифров и их применение.

Воробей Иван Александрович 10701118



Я взял код реализации SDES из лабы 3.

В мейне считываю с консоли ключ, строку, и счетчики в количестве равном длинне строки.



Метод принимает список символов представленных целым числом, ключ и массив счетчиков.

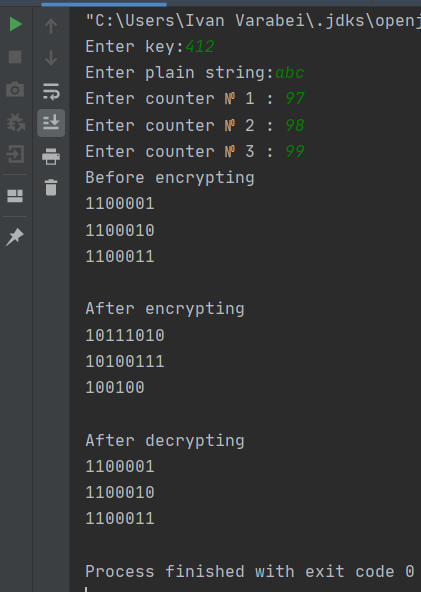
Одна итерация:

1. Шифрую текущий номер счетчика с переданным ключем.
2. Делаю XOR реультата шага 1 и символа plain текста.
3. Добавляю полученный результат в список уже зашифрованных симолов.

static List<Integer> encryptString(List<Integer> plain, int key, int[] counters) {  
 List<Integer> cipher = new ArrayList<>();  
 for (int i = 0; i < plain.size(); i++) {  
 cipher.add(plain.get(i) ^ *sdes*.*facadeEncrypt*(counters[i], key));  
 }  
 return cipher;  
}

Результат выполнения программы. Вводим ключ, plain текст и массив счетчиков.

1. Показываю двоичное представление введенной строки.
2. Показываю результат шифрования.
3. Показываю результат расшифровки в двоичном виде.



**Выводы:** Изучил режимы шифрования. Реализовал режим со счетчиком на языке java.

**ЛИСТИНГ**

**Высокоуровневый класс Counter.java**

public class Counter {  
 private static final SDES *sdes* = new SDES();  
 private static Scanner *in* = new Scanner(System.*in*);  
  
 public static void main(String... sfdg) {  
 int key = *readInt*("Enter key:");  
 String initialString = *readStr*("Enter plain string:");  
 int[] counters = new int[initialString.length()];  
 for(int i = 0; i< counters.length; i++){  
 counters[i] = *readInt*("Enter counter № " + (i + 1) + " : ");  
 }  
 List<Integer> plain = *convertStringToIntList*(initialString);  
 System.*out*.println("Before encrypting");  
 plain.forEach(c -> System.*out*.println(Integer.*toBinaryString*(c)));  
 System.*out*.println();  
  
 List<Integer> cipher = *encryptString*(plain, key, counters);  
 System.*out*.println("After encrypting");  
 cipher.forEach(c -> System.*out*.println(Integer.*toBinaryString*(c)));  
 System.*out*.println();  
  
 System.*out*.println("After decrypting");  
 plain = *encryptString*(cipher, key, counters);  
 plain.forEach(c -> System.*out*.println(Integer.*toBinaryString*(c)));  
 }  
  
 static List<Integer> encryptString(List<Integer> plain, int key, int[] counters) {  
 List<Integer> cipher = new ArrayList<>();  
 for (int i = 0; i < plain.size(); i++) {  
 cipher.add(plain.get(i) ^ *sdes*.*facadeEncrypt*(counters[i], key));  
 }  
 return cipher;  
 }  
  
 static List<Integer> convertStringToIntList(String string) {  
 return string.chars().boxed().collect(Collectors.*toList*());  
 }  
  
 public static int readInt(String str) {  
 System.*out*.print(str);  
 boolean flag = false;  
 while (!flag) {  
 if (!*in*.hasNextInt()) {  
 System.*out*.println("Try again: ");  
 *in*.nextLine();  
 } else {  
 flag = true;  
 }  
 }  
 return *in*.nextInt();  
 }  
  
 public static String readStr(String str) {  
 System.*out*.print(str);  
 return *in*.next();  
 }  
}

**Класс шифрования SDES.java из лабы 3.**

public class SDES {  
 private static final int[][] *S0* = new int[][]{  
 {1, 0, 3, 2},  
 {3, 2, 1, 0},  
 {0, 2, 1, 3},  
 {3, 1, 3, 1}  
 };  
  
 private static final int[][] *S1* = new int[][]{  
 {1, 1, 2, 3},  
 {2, 0, 1, 3},  
 {3, 0, 1, 0},  
 {2, 1, 0, 3}  
 };  
  
 static int facadeEncrypt(int plainSymbol, int key){  
 int p10 = *P10*(key);  
 int lsOnce = *LS*(p10);  
 int lsTwice = *LS*(*LS*(lsOnce));  
 int key1 = *P8*(lsOnce);  
 int key2 = *P8*(lsTwice);  
 return *encrypt*(plainSymbol, new int[]{key1, key2});  
 }  
  
 static int facadeDecrypt(int cipherSymbol, int key){  
 int p10 = *P10*(key);  
 int lsOnce = *LS*(p10);  
 int lsTwice = *LS*(*LS*(lsOnce));  
 int key1 = *P8*(lsOnce);  
 int key2 = *P8*(lsTwice);  
 return *decrypt*(cipherSymbol, new int[]{key1, key2});  
 }  
  
 static int encrypt(int plainSymbol, int[] subKeys) {  
 int result = *IP*(plainSymbol);  
 result = *f*(result, subKeys[0]);  
 result = (result << 28) >>> 24 | (result >>> 4);  
 result = *f*(result, subKeys[1]);  
 return *inverseIP*(result);  
 }  
  
 static int decrypt(int cipherSymbol, int[] subKeys) {  
 int[] newKeys = new int[2];  
 newKeys[0] = subKeys[1];  
 newKeys[1] = subKeys[0];  
 return *encrypt*(cipherSymbol, newKeys);  
 }  
  
 static int f(int plainText, int subKey) {  
 int L = plainText >>> 4;  
 int R = plainText << 28 >>> 28;  
 return (L ^ *F*(R, subKey)) << 4 | R;  
 }  
  
 static int F(int plainText, int subKey) {  
 int permutation = *permutate*(plainText, 3, 0, 1, 2, 1, 2, 3, 0);  
 permutation ^= subKey;  
  
 int substituted = 0;  
 int i = ((permutation & (1 << 7)) >>> 6) | (permutation & (1 << 4)) >>> 4;  
 int j = ((permutation & (1 << 6)) >>> 5) | (permutation & (1 << 5)) >>> 5;  
 substituted |= *S0*[i][j] << 2;  
 i = ((permutation & (1 << 3)) >>> 2) | (permutation & 1);  
 j = ((permutation & (1 << 2)) >>> 1) | (permutation & (1 << 1)) >>> 1;  
 substituted |= *S1*[i][j];  
  
 return *permutate*(substituted, 3, 1, 0, 2);  
 }  
  
 static int P10(int key) {  
 return *permutate*(key, 4, 2, 1, 9, 0, 6, 3, 8, 5, 7);  
 }  
  
 static int LS(int key) {  
 return *permutate*(key, 4, 0, 1, 2, 3, 9, 5, 6, 7, 8);  
 }  
  
 static int P8(int key) {  
 return *permutate*(key, 1, 0, 5, 2, 6, 3, 7, 4);  
 }  
  
 static int IP(int plainText) {  
 return *permutate*(plainText, 1, 3, 0, 4, 7, 5, 2, 6);  
 }  
  
 static int inverseIP(int cryptoText) {  
 return *permutate*(cryptoText, 2, 0, 6, 1, 3, 5, 7, 4);  
 }  
  
 static int permutate(int bits, int... pos) {  
 int permutatedBits = 0;  
 for (int i = 0; i < pos.length; i++) {  
 permutatedBits |= ((bits >>> pos[i]) & 1) << i;  
 }  
 return permutatedBits;  
 }  
}