开发手册

1. 简介

该项目为基于S-Des算法的加密解密开发程序，用于对小规模的简单二进制数据进项加密和解密的操作。通过S-Des算法对数据进行加密，实现信息安全，保障信息传输过程的安全性和可靠性。

本开发手册包含以下内容：

（1）S-Des主要函数：实现代码所用的函数

（2）暴力破解相关函数

通过阅读此开发手册，您可以初步了解项目的主要结构和作用函数。

2.加密解密部分：



2.1密钥扩展

*/\*\*在下述代码中，首先定义了一个长度为10的数组d，通过调用函数p10对输入的密钥k进行置换，并将结果保存在数组d中。 接下来，根据S-DES算法的规则生成两个加密密钥K1和K2。*

*生成K1的过程如下：*

*1.将数组d中的元素按照规则复制到数组k1中。具体操作是，将d中的第2到第5个元素复制到k1的第0到第3个位置，将d中的第7到第10个元素复制到k1的第5到第8个位置。同时，将d中的第1个元素复制到k1的第4个位置，将d中的第6个元素复制到k1的第9个位置。*

*2.将数组k1按照规则进行置换，将结果保存在数组K1中。具体操作是，将k1中的元素按照规则复制到K1中。具体操作是，将k1中的第2到第5个元素复制到K1的第0到第3个位置，将k1中的第0和第1个元素复制到K1的第4和第5个位置，将k1中的第8个元素复制到K1的第6个位置，将k1中的第9个元素复制到K1的第7个位置。*

*类似地，生成K2的过程也是类似的，只是对数组k2的操作稍有不同。 最后，定义了两个函数p10和p8，分别用于进行置换操作。这两个函数的具体实现是，根据给定的置换规则，将输入数组中的元素按照规则复制到输出数组中。*

*这段代码实现了S-DES算法中生成两个子密钥的过程。*

*\*\*/*

int[] d = *p10*(k);  
*//1.1 K1*for(int i=0;i<4;i++){  
 k1[i] = d[i+1];  
 k1[i+5] = d[i+6];  
}  
k1[4] = d[0];  
k1[9] = d[5];  
int[] K1 = *p8*(k1);  
*//1.2 K2*for(int i=0;i<3;i++){  
 k2[i] = d[i+2];  
 k2[i+5] = d[i+7];  
}  
k2[3] = d[0];  
k2[4] = d[1];  
k2[8] = d[5];  
k2[9] = d[6];  
int[] K2 = *p8*(k2);

public static int[] p10(int x[]){  
 int y[] = new int[10];  
 for(int i=0;i<10;i++) {  
 y[i] = x[*P10*[i]-1];  
 }  
 return y;  
}public static int[] p8(int x[]){  
 int y[] = new int[8];  
 for(int i=0;i<8;i++) {  
 y[i] = x[*P8*[i]-1];  
 }  
 return y;  
}

2.2初始置换盒

public static int[] ip(int x[]){  
 int y[] = new int[8];  
 for(int i=0;i<8;i++) {  
 y[i] = x[*IP*[i]-1];  
 }  
 return y;  
}

2.3函数f(k)

*//扩展置换函数（epBox）*

public static int[] epBox(int x[]){  
 int y[] = new int[8];  
 for(int i=0;i<8;i++) {  
 y[i] = x[*EPBox*[i]-1];  
 }  
 return y;  
}

*//S盒函数（sBox*  
public static int[] sBox(int x[]){  
 int z[] = new int[4];  
 int p,q,m;  
 p = x[0]\*2+x[3];  
 q = x[1]\*2+x[2];  
 m = *SBox1*[p][q];  
 if(m==0){  
 z[0]=0;  
 z[1]=0;  
 }  
 if(m==1){  
 z[0]=0;  
 z[1]=1;  
 }  
 if(m==2){  
 z[0]=1;  
 z[1]=0;  
 }  
 if(m==3){  
 z[0]=1;  
 z[1]=1;  
 }  
 p = x[4]\*2+x[7];  
 q = x[5]\*2+x[6];  
 m = *SBox2*[p][q];  
 if(m==0){  
 z[2]=0;  
 z[3]=0;  
 }  
 if(m==1){  
 z[2]=0;  
 z[3]=1;  
 }  
 if(m==2){  
 z[2]=1;  
 z[3]=0;  
 }  
 if(m==3){  
 z[2]=1;  
 z[3]=1;  
 }  
 return z;  
}

*//逆置换函数（spBox）*  
public static int[] spBox(int x[]){  
 int y[] = new int[4];  
 for(int i=0;i<4;i++) {  
 y[i] = x[*SPBox*[i]-1];  
 }  
 return y;  
}

2.4异或函数

public static int[] XOR(int x[],int y[]) {  
 int z[] = new int[8];  
 for (int i = 0; i < x.length; i++) {  
 if (x[i] == y[i]) z[i] = 0;  
 if (x[i] != y[i]) z[i] = 1;  
 }  
 return z;  
}

2.5最终置换盒

public static int[] ip\_1(int x[]){  
 int y[] = new int[8];  
 for(int i=0;i<8;i++) {  
 y[i] = x[*IP\_1*[i]-1];  
 }  
 return y;  
}

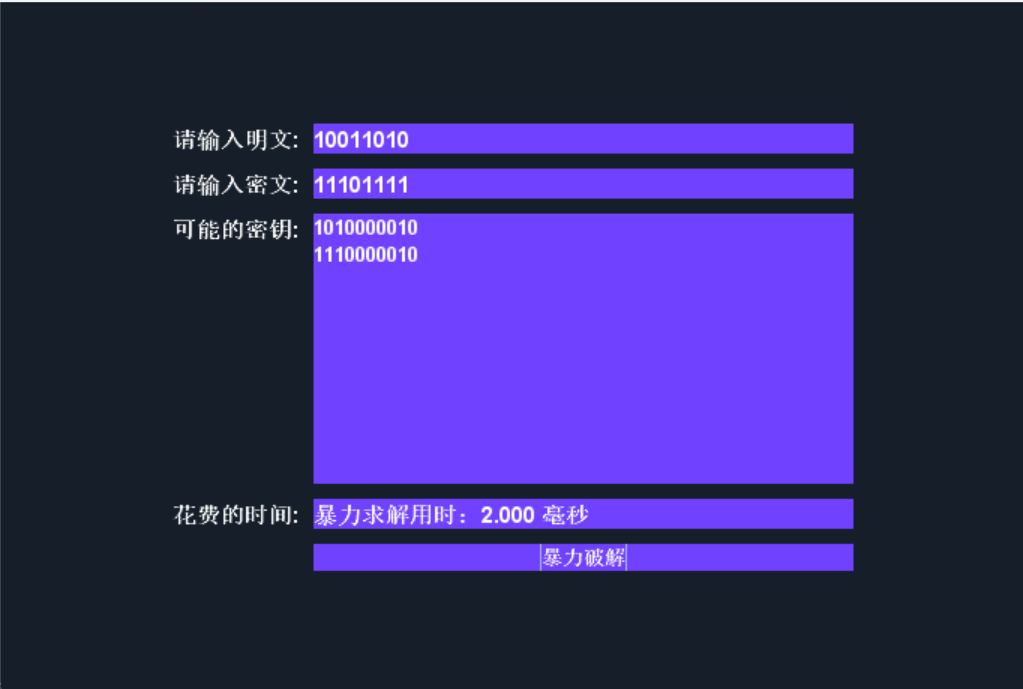
2.6 二进制数加密解密

*// 加密过程*public static String encrypt(String p, String k1, String k2) {  
 *// 初始置换* p = *permute*(p, *IP*);  
 *// 两轮 Fk加密* String l0 = p.substring(0, 4);  
 String r0 = p.substring(4);  
 String l1 = r0;  
 *// 第一轮的 P4* String fResult = *F*(r0, k1, *EP*, *S0*, *S1*, *P4*);  
 *// p4 和 L0 异或* String r1 = String.*format*("%4s", Integer.*toBinaryString*(Integer.*parseInt*(l0, 2) ^ Integer.*parseInt*(fResult, 2))).replace(' ', '0');  
 *// 第二轮的 P4* fResult = *F*(r1, k2, *EP*, *S0*, *S1*, *P4*);  
 *// p4 和 L1 异或* String r2 = String.*format*("%4s", Integer.*toBinaryString*(Integer.*parseInt*(l1, 2) ^ Integer.*parseInt*(fResult, 2))).replace(' ', '0');  
 *// 逆置换* return *permute*(r2 + r1, *IP\_inverse*);  
}  
  
  
*// 解密过程*public static String decrypt(String c, String k1, String k2) {  
 *// 初始置换* c = *permute*(c, *IP*);  
 *// 两轮 Fk解密* String r2 = c.substring(0, 4);  
 String l2 = c.substring(4);  
 *// 第一轮的 P4* String fResult = *F*(l2, k2, *EP*, *S0*, *S1*, *P4*);  
 *// p4 和 R2 异或* String l1 = String.*format*("%4s", Integer.*toBinaryString*(Integer.*parseInt*(r2, 2) ^ Integer.*parseInt*(fResult, 2))).replace(' ', '0');  
 *// 第二轮的 P4* fResult = *F*(l1, k1, *EP*, *S0*, *S1*, *P4*);  
 *// p4 和 R1 异或* String r1 = String.*format*("%4s", Integer.*toBinaryString*(Integer.*parseInt*(l2, 2) ^ Integer.*parseInt*(fResult, 2))).replace(' ', '0');  
 *// 逆置换* return *permute*(r1 + l1, *IP\_inverse*);  
}

2.7 ASCII字符串加密解密

*// 加密函数*public static String encrypt(String plaintext, String key) {  
 StringBuilder ciphertext = new StringBuilder();  
 for (char c : plaintext.toCharArray()) {  
 *// 将明文字符转换为8位二进制字符串* String binaryASCLL = String.*format*("%8s", Integer.*toBinaryString*(c)).replace(' ', '0');  
 *// 使用S-DES算法的加密函数进行加密* String encrypted = SDESGUI.*encrypt*(binaryASCLL, key, key);  
 *// 将加密结果转换为十进制数值* int decimalValue = Integer.*parseInt*(encrypted, 2);  
 *// 将十进制数值转换为对应的ASCII字符* char asciiChar = (char) decimalValue;  
 *// 将加密后的字符添加到密文字符串中* ciphertext.append(asciiChar);  
 }  
 *// 返回密文字符串* return ciphertext.toString();  
}  
  
*// 解密函数*public static String decrypt(String ciphertext, String key) {  
 StringBuilder plaintext = new StringBuilder();  
 for (int i = 0; i < ciphertext.length(); i++) {  
 *// 获取密文字符的ASCII值* char asciiChar = ciphertext.charAt(i);  
 int asciiValue = (int) asciiChar;  
 *// 将ASCII值转换为8位二进制字符串* String binaryASCLL = String.*format*("%8s", Integer.*toBinaryString*(asciiValue)).replace(' ', '0');  
 *// 使用S-DES算法的解密函数进行解密* String decrypted = SDESGUI.*decrypt*(binaryASCLL, key, key);  
 *// 将解密结果添加到明文字符串中* plaintext.append(decrypted);  
 }  
 *// 将明文字符串按照每8位进行分割，将每个二进制字符串转换为对应的十进制数值，再转换为ASCII字符* StringBuilder finalPlaintext = new StringBuilder();  
 for (int i = 0; i < plaintext.length(); i += 8) {  
 String binaryPlaintext = plaintext.substring(i, i + 8);  
 int intValue = Integer.*parseInt*(binaryPlaintext, 2);  
 char charValue = (char) intValue;  
 finalPlaintext.append(charValue);  
 }  
 *// 返回最终的明文字符串* return finalPlaintext.toString();  
}

3.暴力破解：



在这段代码中，通过尝试所有可能的密钥来解密给定的密文，并将解密结果与给定的明文进行比较，以找到正确的密钥。

首先，在从明文输入框和密文输入框获取用户输入的明文和密文之后进行判断，以确保输入的明文和密文是合法的二进制数字，即只包含0和1的两种字符。然后，创建一个线程池，用于并发执行任务。接着创建一个线程安全的列表，用于存储可能的密钥，并记录开始计算暴力求解所用的时间。

之后，循环从0到1023，依次遍历可能的密钥。在每个循环中，代码创建一个新的线程，并将其添加到线程池中。每个线程都会执行以下操作：

- 根据当前的索引值生成一个10位的二进制密钥。

- 使用生成的密钥生成两个子密钥。

- 使用生成的子密钥对密文进行解密。

- 如果解密后的明文与输入的明文相等，则将当前的密钥添加到可能的密钥列表中。

在所有线程执行完毕后，代码记录结束计算暴力求解所用的时间。最后，关闭线程池，并等待所有线程完成。

代码如下：

*// 获取输入的明文和密文*String plaintext = plaintextField.getText();  
String ciphertext = ciphertextField.getText();  
*// 确保输入的明文和密文是合法的二进制数字*if (!plaintext.matches("[01]+") || !ciphertext.matches("[01]+")) {  
 throw new IllegalArgumentException("明文和密文请输入只包含0和1的二进制数字。");  
}  
if (plaintext.length() != 8 || ciphertext.length() != 8) {  
 throw new IllegalArgumentException("明文和密文必须为8位");  
}  
*// 创建线程池*ExecutorService executor = Executors.*newFixedThreadPool*(4);  
*// 存储可能的密钥*List<String> possibleKeys = Collections.*synchronizedList*(new ArrayList<>());  
*// 计算暴力求解所用的时间*long startTime = System.*currentTimeMillis*();  
for (int i = 0; i < 1024; i++) {  
 final int index = i;  
 executor.execute(new Runnable() {  
 @Override  
 public void run() {  
 String key = String.*format*("%10s", Integer.*toBinaryString*(index)).replace(' ', '0');  
 String[] keys = SDESGUI.*generateKey*(key, SDESGUI.*p10Table*, SDESGUI.*p8Table*);  
 String k1 = keys[0];  
 String k2 = keys[1];  
 String guessedPlaintext = SDESGUI.*decrypt*(ciphertext, k1, k2, SDESGUI.*ipTable*, SDESGUI.*epTable*, SDESGUI.*ipNiTable*, SDESGUI.*sbox0*, SDESGUI.*sbox1*, SDESGUI.*p4Table*);  
 if (guessedPlaintext.equals(plaintext)) {  
 possibleKeys.add(key);  
 }  
 }  
 });  
}  
long endTime = System.*currentTimeMillis*();  
executor.shutdown();  
executor.awaitTermination(Long.*MAX\_VALUE*, TimeUnit.*NANOSECONDS*);  
*// 将可能的密钥输出到文本框*StringBuilder keysStringBuilder = new StringBuilder();  
for (String key : possibleKeys) {  
 keysStringBuilder.append(key).append("\n");  
}  
possibleKeysArea.setText(keysStringBuilder.toString());  
long timeElapsed = endTime - startTime;  
String formattedTime = String.*format*("%.3f", (double) timeElapsed); *// 将时间差转换为秒，并保留三位小数*timeTextField.setText("暴力求解用时：" + formattedTime + " 毫秒");