Many Time Pad 实验报告

学号：SA20225172 姓名：郭俊勇

编程语言

- Java

实验目的

- 了解流密码的结构特点；
- 掌握One-time Pad的一般具体实现；
- 通过使用Python（推荐）或者C/C++/java，编程实现一个流密码加密示例的破解，进一步认识在流密码加密中多次使用相同密钥导致的问题

实验内容

- 在掌握流密码结构的基础上，通过本实验观察使用相同流密码密钥加密多个明文导致的严重后果。
- 附件ciphertext.txt有11个十六进制编码的密文，它们是使用流密码加密11个明文的结果，所有密文都使用相同的流密码密钥。
- 实验的目标是解密最后一个密文，并提交明文消息。
- 提示： 对密文进行异或，并考虑当空格与[a~z,A~Z]中的字符进行异或时会发生什么。 2. 附件encrypt.py是用于生成密文的Python示例程序（不影响实验，仅供参考）。

实验原理分析

根据流加密的特点可知：

$$C1 = P1 \oplus K1,\ C2 = P2 \oplus K2.$$

由于本次实验使用了重复的密钥K,因此我们可以合并这两个表达式。

$$C1 \oplus C2 = P1 \oplus K \oplus P2 \oplus K = P1 \oplus P2$$

同时我们可以发现有这样一个规律：

$$space \oplus' a' -' z' =' A' -' Z' space \oplus' A' -' Z' =' a' -' z'$$

由以上规律我们可以知道，如果我们确定密文的某一位的明文是密文(假设C3对应的明文是space)，我们就有如下的表达式：

$$C3 \oplus C4 = space \oplus P4$$

于是我们就可以求出明文P4(结果的大小写有变化，最后统一处理就可以)。

同时我们也可以求出对应的密钥：

$$C3 \oplus space = P3 \oplus K \oplus space = space \oplus K \oplus space = K$$

综合上面的分析，我们需要做的就是找到密文对应的明文为space的位置，然后与目标密文进行异或操作即可得到明文，也可以通过和密钥K异或得到明文。

Java代码实现（源码）

```
public class lab1 {
    public static void main(String[] args) {
        String[] cip = {

 "315c4eeaa8b5f8aaf9174145bf43e1784b8fa00dc71d885a804e5ee9fa40b16349c146fb778cdf
2d3aff021dfff5b403b510d0d0455468aeb98622b137dae857553ccd8883a7bc37520e06e515d22c
954eba5025b8cc57ee59418ce7dc6bc41556bdb36bbca3e8774301fbcaa3b83b220809560987815f
65286764703de0f3d524400a19b159610b11ef3e",

 "234c02ecbbfbafa3ed18510abd11fa724fcda2018a1a8342cf064bbde548b12b07df44ba7191d9
606ef4081ffde5ad46a5069d9f7f543bedb9c861bf29c7e205132eda9382b0bc2c5c4b45f919cf3a
9f1cb74151f6d551f4480c82b2cb24cc5b028aa76eb7b4ab24171ab3cdadb8356f",

 "32510ba9a7b2bba9b8005d43a304b5714cc0bb0c8a34884dd91304b8ad40b62b07df44ba6e9d8a
2368e51d04e0e7b207b70b9b8261112bacb6c866a232dfe257527dc29398f5f3251a0d47e503c66e
935de81230b59b7afb5f41afa8d661cb",

 "32510ba9aab2a8a4fd06414fb517b5605cc0aa0dc91a8908c2064ba8ad5ea06a029056f47a8ad3
306ef5021eafe1ac01a81197847a5c68a1b78769a37bc8f4575432c198ccb4ef63590256e305cd3a
9544ee4160ead45aef520489e7da7d835402bca670bda8eb775200b8dabbba246b130f040d8ec644
7e2c767f3d30ed81ea2e4c1404e1315a1010e7229be6636aaa",

 "3f561ba9adb4b6ebec54424ba317b564418fac0dd35f8c08d31a1fe9e24fe56808c213f17c81d9
607cee021dafe1e001b21ade877a5e68bea88d61b93ac5ee0d562e8e9582f5ef375f0a4ae20ed86e
935de81230b59b73fb4302cd95d770c65b40aaa065f2a5e33a5a0bb5dcaba43722130f042f8ec85b
7c2070",

 "32510bfbacfbb9befd54415da243e1695ecabd58c519cd4bd2061bbde24eb76a19d84aba34d8de
287be84d07e7e9a30ee714979c7e1123a8bd9822a33ecaf512472e8e8f8db3f9635c1949e640c621
854eba0d79eccf52ff111284b4cc61d11902aebc66f2b2e436434eacc0aba938220b084800c2ca4e
693522643573b2c4ce35050b0cf774201f0fe52ac9f26d71b6cf61a711cc229f77ace7aa88a2f199
83122b11be87a59c355d25f8e4",

 "32510bfbacfbb9befd54415da243e1695ecabd58c519cd4bd90f1fa6ea5ba47b01c909ba7696cf
606ef40c04afe1ac0aa8148dd066592ded9f8774b529c7ea125d298e8883f5e9305f4b44f915cb2b
d05af51373fd9b4af511039fa2d96f83414aaaf261bda2e97b170fb5cce2a53e675c154c0d968159
6934777e2275b381ce2e40582afe67650b13e72287ff2270abcf73bb028932836fbdecfecee0a3b8
94473c1bbeb6b4913a536ce4f9b13f1efff71ea313c8661dd9a4ce",

 "315c4eeaa8b5f8bffd11155ea506b56041c6a00c8a08854dd21a4bbde54ce56801d943ba708b8a
3574f40c00fff9e00fa1439fd0654327a3bfc860b92f89ee04132ecb9298f5fd2d5e4b45e40ecc3b
9d59e9417df7c95bba410e9aa2ca24c5474da2f276baa3ac325918b2daada43d6712150441c2e04f
6565517f317da9d3",

 "271946f9bbb2aeadec111841a81abc300ecaa01bd8069d5cc91005e9fe4aad6e04d513e96d99de
2569bc5e50eeeca709b50a8a987f4264edb6896fb537d0a716132ddc938fb0f836480e06ed0fcd6e
9759f40462f9cf57f4564186a2c1778f1543efa270bda5e933421cbe88a4a52222190f471e9bd15f
652b653b7071aec59a2705081ffe72651d08f822c9ed6d76e48b63ab15d0208573a7eef027",

 "466d06ece998b7a2fb1d464fed2ced7641ddaa3cc31c9941cf110abbf409ed39598005b3399ccf
afb61d0315fca0a314be138a9f32503bedac8067f03adbf3575c3b8edc9ba7f537530541ab0f9f3c
d04ff50d66f1d559ba520e89a2cb2a83"
        };
        String target =
"32510ba9babebbbefd001547a810e67149caee11d945cd7fc81a05e9f85aac650e9052ba6a8cd82
57bf14d13e6f0a803b54fde9e77472dbff89d71b57bddef121336cb85ccb8f3315f4b52e301d16e9
f52f904";
```

```
        //将给定的是个密文进行截断，因为我们只需要翻译target目标长度的字符串，只需要这么多位
        for (int i = 0; i < cip.length; i++) {
            cip[i] = cip[i].substring(0, target.length());

        }

        //创建记录的Map
        Map<Integer, Character> result = new HashMap<>();
        //创建记录key的Map
        Map<Integer, String> key = new HashMap<>();
        //遍历10个数组,寻找space
        for (int i = 0; i < cip.length; i++) {
            for (int j = 0; j < target.length(); j += 2) {
                //获得每个字符的16进制
                BigInteger temp1 = new BigInteger(cip[i].substring(j, j + 2),
16);
                //设置一个标志位记录可能为空格的个数
                int count = 0;
                for (int k = 0; k < cip.length; k++) {
                    BigInteger temp2 = new BigInteger(cip[k].substring(j, j +
2), 16);
                    if (IsCharacter((char) temp1.xor(temp2).intValue())) {
                        count++;
                    }
                    if (count > 4) {
                        BigInteger temp3 = new BigInteger(target.substring(j, j
+ 2), 16);
                        //空格
                        BigInteger space = new BigInteger("20", 16);
                        //与space异或得到key
                        BigInteger xor = temp1.xor(space);
                        String s = xor.toString(16);
                        //密文与K异或得到明文
                        char temp4 = (char) temp3.xor(xor).intValue();
                        key.put(j, s);
                        if (IsCharacter(temp4)) {
                            result.put(j, temp4);
                            break;
                        } else {
                            continue;
                        }
                    }
                }
            }
        }

        //输出key
        String key1 = "";
        for (int i = 0; i < target.length(); i += 2) {
            key1 = key1 + key.get(i);

        }
        System.out.println("密钥key:");
        System.out.println(key1);

        //输出明文
        String s = "";
        for (int i = 0; i < target.length(); i++) {
```

```
            //存储时是间隔存储所以需要删除null的位置
            if (result.get(i) == null && result.get(i + 1) == null) {
                s += " ";
                i++;
            } else if (result.get(i) != null) {
                s += result.get(i).toString().toLowerCase();
            }
        }
        System.out.println("明文plaintext：");
        System.out.println(s);


    }

    //判断是否是字符a-z,A-Z,标点符号
    private static boolean IsCharacter(char c) {
        if (c >= 65 && c <= 90) {
            return true;
        }
        if (c >= 97 && c < 122) {
            return true;
        }
        if (c == 44 || c == 58) {
            return true;
        }

        return false;
    }
}
```

运行结果