密码学原理
实验一：伪随机与一次一密

学号：2022113564　　姓名：张哲恺

实验目的：本实验旨掌握伪随机生成工具使用、一次一密加密应用，以及多次加密（使用相同密钥多次加密不同消息）破解方法。

1、 伪随机生成工具

(1) 在 Linux 命令行中生成随机串

要求：至少 2 种方法，工具原理描述（熵来源），程序运行截图

1：使用 Linux uuid，uuid 是通用唯一标识符，是一种标准化的格式，其目的是让分布式系统中的所有元素，都能有唯一的辨识信息。uuid 由 32 位数字组成，编码采用十六进制。1~8 位采用系统时间，在系统时间上精确到毫秒级保证时间上的唯一性；9~16 位采用底层的 IP 地址，在服务器集群中的唯一性；17~24 位采用当前对象的 HashCode 值，在一个内部对象上的唯一性；25~32 位采用调用方法的一个随机数，在一个对象内的毫秒级的唯一性。使用命令行随机生成一个 uuid 可保证时间空间上的唯一性。

```
zigo@zigo-virtual-machine:~$ for i in {1..5}
> do
> echo $i $(cat /proc/sys/kernel/random/uuid)
> done
1 3bd8385b-990f-4029-9cd8-2f71af1c8a01
2 aef53fed-b9ca-451b-9267-a0c34fcdb650
3 d9f09add-d99c-498f-9f7f-dbd609982996
4 ad1a1cf7-21f4-4db2-8615-5f79ecb20ad6
5 b69ba38c-1d6f-43b8-a1e8-e8042e750412
zigo@zigo-virtual-machine:~$
```

2：使用 date +%N%s 通过 Linux 的时间戳来获取随机数。%s 为获取秒级别的时间戳；%N 为获取纳秒级别的时间戳。使用%s%N 组合随机数，重复概率大大降低但是生成的随机数在较短时间内可能较为接近，熵来源较低。

```
zigo@zigo-virtual-machine:~/桌面$ date +%N%s
73605691417108481l8
```

(2) 调用密码程序库接口生成随机串

要求：至少 2 种方法，库与接口描述，代码截图，程序运行截图

1：调用 openssl 库。rand 代码用于生成随机字符，-base64 指使用 base64 编码，-hex 指使用十六进制格式，8 指生成随机串占 8 个字节，md5sum 使用哈希计算其 md5 哈希值，最后即可得到随机字符串。

```
zigo@zigo-virtual-machine:~$ for i in {1..5}
> do
> echo $i $(openssl rand -base64 -hex 8 | md5sum)
> done
1 5f956b3a310ddbf5b72fce59dac89219 -
2 669bb0a92ae32ca677b33ce92582b7ad -
3 d315eb9717135ef3b27c0f4d33ffc4db -
4 237b2ef71e9c55a275d8ef9c74b1441e -
5 4db4463cce393b2d190ddbd1db70da5e -
zigo@zigo-virtual-machine:~$
```

2：使用 pwgen 调用密码库的接口，-s 代表生成绝对随机的字符串，-y 代表生成的字符串包含特殊符号，16 代表生成字符串的长度，1 代表生成的密码个数。

```
zigo@zigo-virtual-machine:~/桌面$ pwgen -s -y 16 1
_g]1K(,qNL;WGcb|
zigo@zigo-virtual-machine:~/桌面$ pwgen -s 16 1
Ik2heanEP3jiGFPJ
```

```python
import pwgen
import secrets

def generate1(length):
    random_string = pwgen.pwgen(length, capitalize=True, no_symbols=True)
    return random_string

random_string = generate1(10)
print("pwgen_gen: ",random_string)

def generate2(length):
    random_string = secrets.token_urlsafe(length)[:length]
    return random_string

random_string = generate2(10)
print("token_urlsafe_gen: ",random_string)
```

```
pwgen_gen:  jBHMKgFAx7
token_urlsafe_gen:  n4xAUfVN8a
```

方法一使用 pwgen.pwgen()函数来生成指定长度随机串，”capitalize=True 表示生成随机串中字母为大写，no_symbols=True 表示随机串不包含特殊字符。

方法二使用 secrets 模块中的 secrets.token_urlsafe()函数生成 URL 安全的随即字节，返回 Base64 编码的字符串。

2、 应用伪随机生成工具进行一次一密加密

要求：加密方案描述，代码截图，程序运行截图

```python
import secrets

def generate_key(plaintext) :
    key = secrets.token_hex(len(plaintext))
    return key

def encrypt(plaintext , key) :
    ciphertext = ""
    for i in range(int(len(plaintext)/2)) :
        int1 = int(plaintext[i * 2 : i * 2 + 2] , 16)
        int2 = int(key[i * 2 : i * 2 + 2] , 16)
        ciphertext += format(int1 ^ int2 , '02x')
    return ciphertext

def decrypt(ciphertext , key) :
    plaintext = ""
    for i in range(int(len(ciphertext)/2)) :
        int1 = int(ciphertext[i * 2 : i * 2 + 2] , 16)
        int2 = int(key[i * 2 : i * 2 + 2] , 16)
        plaintext += chr(int1 ^ int2)
    return plaintext



plaintext = "Hello World!"
key = generate_key(plaintext.encode().hex())
print("key: " , key)
ciphertext = encrypt(plaintext.encode().hex() , key)
print("ciphertext: " , ciphertext)
print("plaintext: " , decrypt(ciphertext , key))
```

```
key:  b2d1dce5a8b0e469d9cca6d46bf5f3a09214f1e873507950
ciphertext:  fab4b089c790b306aba0c2f5
plaintext:  Hello World!
```

首先 generate_key()生成一个与明文等长的随机密钥， encrypt()使用随机密钥通过逐位加密明文， decrypt()解密密文。

3、分析多次加密（Many-Time-Pad）

多次使用相同的密钥加密不同明文在窃听攻击下是不安全的。当加密次数足够多时，获得足够多的密文，根据明文特点足以破解。请对以下使用相同的密钥加密的 11 个十六进制编码密文进行分析，并破解最后一个密文。明文是普通的 ASCII 编码英文，有空格，2 个 16 进制数字对应一个英文字符。

提示：将密文进行**异或**运算将得到明文的异或结果。考虑在明文的一个空格与一个英文字符[a-zA-Z]异或会得到什么样的密文异或结果。本题给出的密文“足够多”，多到对应明文中的空格出现在了各个位置。

密文#1:

315c4eeaa8b5f8aaf9174145bf43e1784b8fa00dc71d885a804e5ee9fa40b16349c146f
b778cdf2d3aff021dfff5b403b510d0d0455468aeb98622b137dae857553ccd8883a7b
c37520e06e515d22c954eba5025b8cc57ee59418ce7dc6bc41556bdb36bbca3e87743
01fbcaa3b83b220809560987815f65286764703de0f3d524400a19b159610b11ef3e

密文#2:

234c02ecbbfbafa3ed18510abd11fa724fcda2018a1a8342cf064bbde548b12b07df44
ba7191d9606ef4081ffde5ad46a5069d9f7f543bedb9c861bf29c7e205132eda9382b0
bc2c5c4b45f919cf3a9f1cb74151f6d551f4480c82b2cb24cc5b028aa76eb7b4ab2417
1ab3cdadb8356f

密文#3:

32510ba9a7b2bba9b8005d43a304b5714cc0bb0c8a34884dd91304b8ad40b62b07df
44ba6e9d8a2368e51d04e0e7b207b70b9b8261112bacb6c866a232dfe257527dc2939
8f5f3251a0d47e503c66e935de81230b59b7afb5f41afa8d661cb

密文#4:

32510ba9aab2a8a4fd06414fb517b5605cc0aa0dc91a8908c2064ba8ad5ea06a02905
6f47a8ad3306ef5021eafe1ac01a81197847a5c68a1b78769a37bc8f4575432c198ccb
4ef63590256e305cd3a9544ee4160ead45aef520489e7da7d835402bca670bda8eb77
5200b8dabbba246b130f040d8ec6447e2c767f3d30ed81ea2e4c1404e1315a1010e72
29be6636aaa

密文#5:

3f561ba9adb4b6ebec54424ba317b564418fac0dd35f8c08d31a1fe9e24fe56808c213

f17c81d9607cee021dafe1e001b21ade877a5e68bea88d61b93ac5ee0d562e8e9582f5ef375f0a4ae20ed86e935de81230b59b73fb4302cd95d770c65b40aaa065f2a5e33a5a0bb5dcaba43722130f042f8ec85b7c2070

密文#6:

32510bfbacfbb9befd54415da243e1695ecabd58c519cd4bd2061bbde24eb76a19d84aba34d8de287be84d07e7e9a30ee714979c7e1123a8bd9822a33ecaf512472e8e8f8db3f9635c1949e640c621854eba0d79eccf52ff111284b4cc61d11902aebc66f2b2e436434eacc0aba938220b084800c2ca4e693522643573b2c4ce35050b0cf774201f0fe52ac9f26d71b6cf61a711cc229f77ace7aa88a2f19983122b11be87a59c355d25f8e4

密文#7:

32510bfbacfbb9befd54415da243e1695ecabd58c519cd4bd90f1fa6ea5ba47b01c909ba7696cf606ef40c04afe1ac0aa8148dd066592ded9f8774b529c7ea125d298e8883f5e9305f4b44f915cb2bd05af51373fd9b4af511039fa2d96f83414aaaf261bda2e97b170fb5cce2a53e675c154c0d9681596934777e2275b381ce2e40582afe67650b13e72287ff2270abcf73bb028932836fbdecfecee0a3b894473c1bbeb6b4913a536ce4f9b13f1efff71ea313c8661dd9a4ce

密文#8:

315c4eeaa8b5f8bffd11155ea506b56041c6a00c8a08854dd21a4bbde54ce56801d943ba708b8a3574f40c00fff9e00fa1439fd0654327a3bfc860b92f89ee04132ecb9298f5fd2d5e4b45e40ecc3b9d59e9417df7c95bba410e9aa2ca24c5474da2f276baa3ac325918b2daada43d6712150441c2e04f6565517f317da9d3

密文#9:

271946f9bbb2aeadec111841a81abc300ecaa01bd8069d5cc91005e9fe4aad6e04d513e96d99de2569bc5e50eeeca709b50a8a987f4264edb6896fb537d0a716132ddc938fb0f836480e06ed0fcd6e9759f40462f9cf57f4564186a2c1778f1543efa270bda5e933421cbe88a4a52222190f471e9bd15f652b653b7071aec59a2705081ffe72651d08f822c9ed6d76e48b63ab15d0208573a7eef027

密文#10:

466d06ece998b7a2fb1d464fed2ced7641ddaa3cc31c9941cf110abbf409ed39598005

b3399ccfafb61d0315fca0a314be138a9f32503bedac8067f03adbf3575c3b8edc9ba7f537530541ab0f9f3cd04ff50d66f1d559ba520e89a2cb2a83

目标密文 (解密此密文):

32510ba9babebbbefd001547a810e67149caee11d945cd7fc81a05e9f85aac650e9052ba6a8cd8257bf14d13e6f0a803b54fde9e77472dbff89d71b57bddef121336cb85ccb8f3315f4b52e301d16e9f52f904

要求：描述解密原理，代码截图，程序运行截图

```
cipherText = [
    "315c4eeaa8b5f8aaf9174145bf43e1784b8fa00dc71d885a804e5ee9fa40b16349c146fb778cdf2d3aff021dfff5b403b510d0d0455468aeb98622b137dae857553ccd8883a7bc37520e06e515d22c954eba5025b8cc57ee59418ce7dc6bc41556bdb36bbca3e8774301fbcaa3b83b220809560987815f65286764703de0f3d524400a19b159610b11ef3e",
    "234c02ecbbfbafa3ed18510abd11fa724fcda2018a1a8342cf064bbde548b12b07df44ba7191d9606ef4081ffde5ad46a5069d9f7f543bedb9c861bf29c7e205132eda9382b0bc2c5c4b45f919cf3a9f1cb74151f6d551f4480c82b2cb24cc5b028aa76eb7b4ab24171ab3cdadb8356f",
    "32510ba9a7b2bba9b8005d43a304b5714cc0bb0c8a34884dd91304b8ad40b62b07df44ba6e9d8a2368e51d04e0e7b207b70b9b8261112bacb6c866a232dfe257527dc29398f5f3251a0d47e503c66e935de81230b59b7afb5f41afa8d661cb",
    "32510ba9aab2a8a4fd06414fb517b5605cc0aa0dc91a8908c2064ba8ad5ea06a029056f47a8ad3306ef5021eafe1ac01a81197847a5c68a1b78769a37bc8f4575432c198ccb4ef63590256e305cd3a9544ee4160ead45aef520489e7da7d835402bca670bda8eb775200b8dabbba246b130f040d8ec6447e2c767f3d30ed81ea2e4c1404e1315a1010e7229be6636aaa",
    "3f561ba9adb4b6ebec54424ba317b564418fac0dd35f8c08d31a1fe9e24fe56808c213f17c81d9607cee021dafe1e001b21ade877a5e68bea88d61b93ac5ee0d562e8e9582f5ef375f0a4ae20ed86e935de81230b59b73fb4302cd95d770c65b40aaa065f2a5e33a5a0bb5dcaba43722130f042f8ec85b7c2070",
    "32510bfbacfbb9befd54415da243e1695ecabd58c519cd4bd2061bbde24eb76a19d84aba34d8de287be84d07e7e9a30ee714979c7e1123a8bd9822a33ecaf512472e8e8f8db3f9635c1949e640c621854eba0d79eccf52ff111284b4cc61d11902aebc66f2b2e436434eacc0aba938220b084800c2ca4e693522643573b2c4ce35050b0cf774201f0fe52ac9f26d71b6cf61a711cc229f77ace7aa88a2f19983122b11be87a59c355d25f8e4",
    "32510bfbacfbb9befd54415da243e1695ecabd58c519cd4bd90f1fa6ea5ba47b01c909ba7696cf606ef40c04afe1ac0aa8148dd066592ded9f8774b529c7ea125d298e8883f5e9305f4b44f915cb2bd05af51373fd9b4af511039fa2d96f83414aaaf261bda2e97b170fb5cce2a53e675c154c0d9681596934777e2275b381ce2e40582afe67650b13e72287ff2270abcf73bb028932836fbdecfecee0a3b894473c1bbeb6b4913a536ce4f9b13f1efff71ea313c8661dd9a4ce",
    "315c4eeaa8b5f8bffd11155ea506b56041c6a00c8a08854dd21a4bbde54ce56801d943ba708b8a3574f40c00fff9e00fa1439fd0654327a3bfc860b92f89ee04132ecb9298f5fd2d5e4b45e40ecc3b9d59e9417df7c95bba410e9aa2ca24c5474da2f276baa3ac325918b2daada43d6712150441c2e04f6565517f317da9d3",
    "271946f9bbb2aeadec111841a81abc300ecaa01bd8069d5cc91005e9fe4aad6e04d513e96d99de2569bc5e50eeeca709b50a8a987f4264edb6896fb537d0a716132ddc938fb0f836480e06ed0fcd6e9759f40462f9cf57f4564186a2c1778f1543efa270bda5e933421cbe88a4a52222190f471e9bd15f652b653b7071aec59a2705081ffe72651d08f822c9ed6d76e48b63ab15d0208573a7eef027",
```

```
    "466d06ece998b7a2fb1d464fed2ced7641ddaa3cc31c9941cf110abbf409ed39598005b3399ccfafb61d
0315fca0a314be138a9f32503bedac8067f03adbf3575c3b8edc9ba7f537530541ab0f9f3cd04ff50d66f1d5
59ba520e89a2cb2a83",
    "32510ba9babebbbefd001547a810e67149caee11d945cd7fc81a05e9f85aac650e9052ba6a8cd8257bf1
4d13e6f0a803b54fde9e77472dbff89d71b57bddef121336cb85ccb8f3315f4b52e301d16e9f52f904"
]

def xor(ciphertext):
    xor_result = [["" for _ in range(11)] for _ in range(11)]
    for i in range(11):
        for j in range(11):
            result = ""
            for k in range(int(len(ciphertext[10])/2)) :
                int1 = int(ciphertext[i][k * 2 : k * 2 + 2], 16)
                int2 = int(ciphertext[j][k * 2 : k * 2 + 2], 16)
                result += format(int1 ^ int2, '02x')
            xor_result[i][j] = result
    return xor_result

def hex_to_ascii(xor_result):
    xor_ascii_result = [["" for _ in range(11)] for _ in range(11)]
    for i in range(11):
        for j in range(11):
            ascii_string = ""
            for k in range(int(len(cipherText[10])/2)) :
                ascii_char = chr(int(xor_result[i][j][k * 2 : k * 2 + 2], 16))
                if ('a' <= ascii_char <= 'z' or 'A' <= ascii_char <= 'Z'):
                    ascii_string += ascii_char
                else:
                    ascii_string += '-'
            xor_ascii_result[i][j] = ascii_string
            print(i, " ", j, " :", ascii_string)
        print("\n")
    return xor_ascii_result

def cal_space(ascii_result) :
    key = ""
    for k in range(int(len(cipherText[10])/2)) :
        flag = 0
        for i in range(11) :
            if (flag == 1) :
                break
            cal = 0
            for j in range(11) :
```

```
                if ('a' <= ascii_result[i][j][k] <= 'z' or 'A' <= ascii_result[i][j][k] <=
'Z') :
                    cal += 1
            if (cal >= 6) :
                print(k , " " , i , "\n")
                int1 = int(cipherText[i][k * 2 : k * 2 + 2] , 16)
                key += format(int1 ^ 0x20 , '02x')
                flag =1

    print("key: " , key)
    return key

def decode(key) :
    plaintext = " "
    for i in range(int(len(cipherText[10]) / 2)) :
        int1 = int(cipherText[10][i * 2 : i * 2 + 2] , 16)
        int2 = int(key[i * 2 : i * 2 + 2] , 16)
        plaintext16 = format(int1 ^ int2 , '02x')
        plaintext += chr(int(plaintext16 , 16))
    print(plaintext)
    return plaintext

xor_result = xor(cipherText)
xor_ascii_result = hex_to_ascii(xor_result)
key = cal_space(xor_ascii_result)
plaintext = decode(key)
```

第一步先将全部的密文串互相异或得到以下这样的字符串

```
0   0  : ------------------------------------------------------------------------------
0   1  : --L--NW----O-R---B--M---OH-T---HN--A---MT------E--MO--SC-NC----RF--------EC-----R--
0   2  : --EC--C-A----GT--O--M---Y-ZQW--HN--A--U-R---------KR-EC--ND------A---RO-H-A---B--RB
0   3  : --EC--P------TT--O-----RBH-AW---KQ------T---P-----GT------K-L--------O-ST--P------T-
0   4  : --UC--NA-C---TT------B-RSTA---T-A-U----MF---P-T----W------C----Z--C--RS---L---B--RB
0   5  : --E--NA--C-------E-U--E-RHET----P--ACT--A-O-----R-GL-EK--------E--C---ET--O-U------
0   6  : --E--NA--C-------E-U--E-YAAO----H-OA---MT---P---------EC--V----E--C--RU--EB----E-OC
0   7  : ----------T--ET--I--M---RT-T--T-H--A--U-N-----T--SO---O--NB--S-SF----RA--EC------S-
0   8  : -E----V---Y--Y-HEE------I-------M-U-----SC-M------ZH---C--M---OAF-----D-------B--NT
0   9  : w-H-A-O-----Ro---R------O-TR-I-Z-ACHN--------U----ZOw-SC--EA------CT--I---GN-M-E-O-
0   10  : --EC--C---T--S---EN--XE-HT------GQ-A----A-O--------N--E-A-S-L--EF---O-O--ET---B--CT


1   0  : --L--NW----O-R---B--M---OH-T---HN--A---MT------E--MO--SC-NC----RF--------EC-----R--
1   1  : ------------------------------------------------------------------------------
1   2  : ---E-I--U--I--O-----------O-H---------SC-------A-----E-A-------RAS---EO-FF----T-A-S
1   3  : ---E-I-----E--O-----C--J----H--A-O-N---P----R--G------SL-O--R--RG---N-SO-I------XY-
1   4  : ---E-O-H-L-A--O--B--YE-J--TT--TC--WK--------R-MG--C---SS-E------E-T--ES--A----T-A-S
1   5  : ---------L-W-R-----YO-N---P----A----EI-H--E----HB----E-E-PC-----T-T---EO-R--Y---R-L
1   6  : ---------L-W-R-----YO-N---T----P--M---------R--L---O-----O------N-T--EU--------OFBR
1   7  : --L--NW---DT--O---------------TC------SU------MI-E-O---N-----N-------EA---------E--
1   8  : -UD--I----IK--FBA---R-----NT---E--WS---E-HVO---O---------A----E-------D--EC---T-ECE
1   9  : e---Rc-----EP-------I-----A--A----A-H--------E-R----M----H-O---RO-TO--I--N-R-P-OSBL
1   10  : ---E-E----DM------L-S-N---NT---N-O-----E--E----E-IC----RAU--R-------N-O-------T-NNE


2   0  : --EC--C-A----GT--O--M---Y-ZQW--HN--A--U-R---------KR-EC--ND------A---RO-H-A---B--RB
2   1  : ---E-I--U--I--O-----------O-H---------SC-------A-----E-A-------RAS---EO-FF----T-A-S
2   2  : ------------------------------------------------------------------------------
2   3  : --------E-----------C--E--O----A-O-N--Y-----O--------MC--O--I----O--TA-FC-----T---S
2   4  : -------BTT-------O--Yk-E---QO-SC--WK--SC----O-R---E--OC--E-----Z-SL-----E----------
2   5  : ---R-I--ET---GT----TO-E-----O--A----ZET---P-----P--------PD----E-SL--F-FF---C-O--R-
2   6  : ---R-I--ET---GT----TO-E-----G--P--M---EC----O------R-H-A-O-----E-TL-----EF----EC---
2   7  : --EC--C-E-H---------------O-H-SC--------------R--H-R-R-------V-SAS------DF----U---S
2   8  : -HMP----T-E----AB---R------QS--E--WS--T--YCT---------SOA-A----EAAP---E--R-A--------
2   9  : t--EN---C---N-X-----I-------YI----A-W-E------G------SA-A-H-R-----FLO-R--I--N-YRC---
2   10  : --------E-H---S---U-SqE----QU--N-O----R---P------DE--V--NU--I--EAK--TM--EF---------
```

第二步观察每个密文串与其他十个密文串异或得到的 10 个结果，数 10 个结果中同一位置为大小写字母的个数，寻找个数大于 6 的情况记录下位置和密文编号得到如下表。

0	9
1	8
2	0
3	2
4	9
5	1
6	0
7	4
8	2

由于大小写字母在 ASCII 编码中正好相差 32，而空格的 ASCII 编码就是 32，因此大写字母与空格异或得到的是小写字母，同理小写字母异或之后会得到大写字母，据此我们可以大胆推测，记录下来的这些位置是明文中空格的位置。实际上这些位置可能有偏差，两个字符只要满足部分条件，得到的也有可能是大小写字母。

第三步，从第一位开始，将记录下来的对应编号的位置的密文与空格异或，由于已知明文异或密钥后得到密文，且明文推测为空格，因此将密文与空格异或将得到密文。将得到的所有密钥拼接起来即为所求密钥。

```
key:  66396e89c9dbd8cb9874352acd6395102eafce78aa7fed28a06e6bc98d29c50b69b033db14f8aa401a9c6d708f80c066c763f0f0123148cdd8e802d05ba98777335daefcecd59c433a6b268b60bf4ef03c9a61
```

第四步，将密钥与密文异或解密得到明文。

```
The secuet message is: Whtn using aa~tream cipher,.never use the key more than once
```

可以看到该明文不是真正的明文，由于上述得到的空格位置只是推测，因此存在个别字母解密错误，接着我们用该密钥解密其他密文，得到以下结果。

```
0  We can aactor the number  5 with qu ctum computers  We can also factor the number 1
1  Euler whuld probably enjoh that nowaeis theorem bemomes a corner stone of crypto -
2  The nicb thing about Keey}oq is nowaze cryptographkrs can drive a lot of fancy cars
3  The cipoertext produced bh a weak e/nryption algorgthm looks as good as ciphertext
4  You don t want to buy a stt of car *hys from a guy.who specializes in stealing cars
5  There aue two types of crhptographya  that which wgll keep secrets safe from your l
6  There aue two types of cyatography:abne that allow} the Government to use brute for
7  We can tee the point whert the chipads unhappy if o wrong bit is sent and consumes
8  A (privfte-key)  encrypti~n scheme 2yates 3 algorizhms, namely a procedure for gene
9   The Coicise OxfordDictioary (2006h-deī¬nes crypzo as the art of  writing o r sol
10  The secuet message is: Whtn using aa~tream cipher,.never use the key more than once
```

我们对编号 10 的解密结果进行人工校验：

The secret message is: When using a stream cipher, never use the key more than once

得到的明文再与编号 10 的密文进行异或得到修正后的密钥，并用新的密钥解密其他的密文，得到如下结果

新的密钥：

```
new_key 66396e89c9dbd8cc9874352acd6395102eafce78aa7fed28a07f6bc98d29c50b69b0339a19f8aa401a9c6d708f80c066c763fef0123148cdd8e802d05ba98777335daefcecd59c433a6b268b60bf4ef03c9a61
```

解密结果：

```
We can factor the number 15 with quantum computers. We can also factor the number 1
Euler would probably enjoy that now his theorem becomes a corner stone of crypto -
The nice thing about Keeyloq is now we cryptographers can drive a lot of fancy cars
The ciphertext produced by a weak encryption algorithm looks as good as ciphertext
You don't want to buy a set of car keys from a guy who specializes in stealing cars
There are two types of cryptography - that which will keep secrets safe from your l
There are two types of cyptography: one that allows the Government to use brute for
We can see the point where the chip is unhappy if a wrong bit is sent and consumes
A (private-key)  encryption scheme states 3 algorithms, namely a procedure for gene
 The Concise OxfordDictionary (2006) deī¬nes crypto as the art of  writing o r sol
The secret message is: When using a stream cipher, never use the key more than once
```

在密钥长度范围内，得到的结果是可读的，因此我们认为新的密钥是正确的，故目标密文的揭秘结果为

The secret message is: When using a stream cipher, never use the key more than once