**密码学原理**

**实验一：伪随机与一次一密**

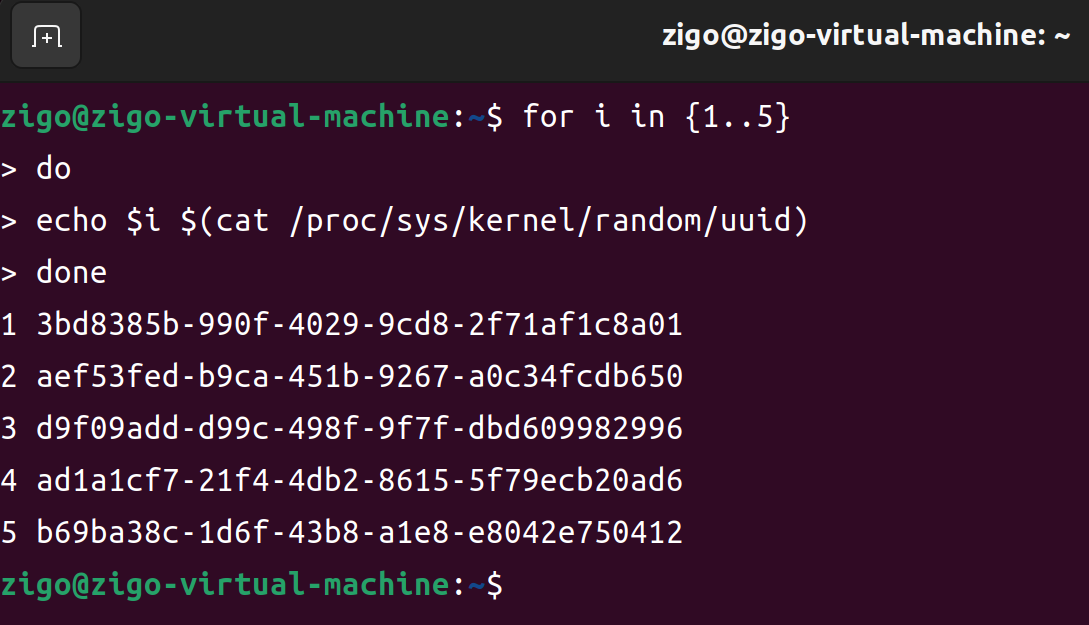
学号：2022113564 姓名：张哲恺

**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。

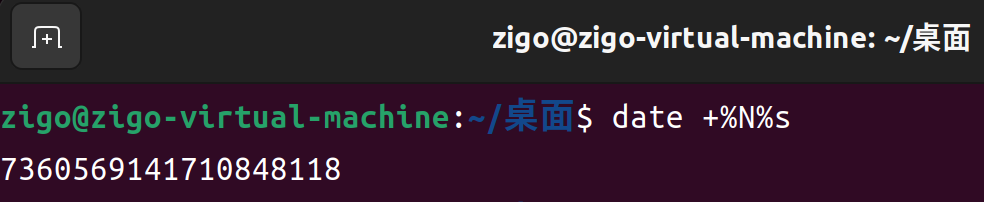
1. **伪随机生成工具**
2. 在Linux命令行中生成随机串

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。



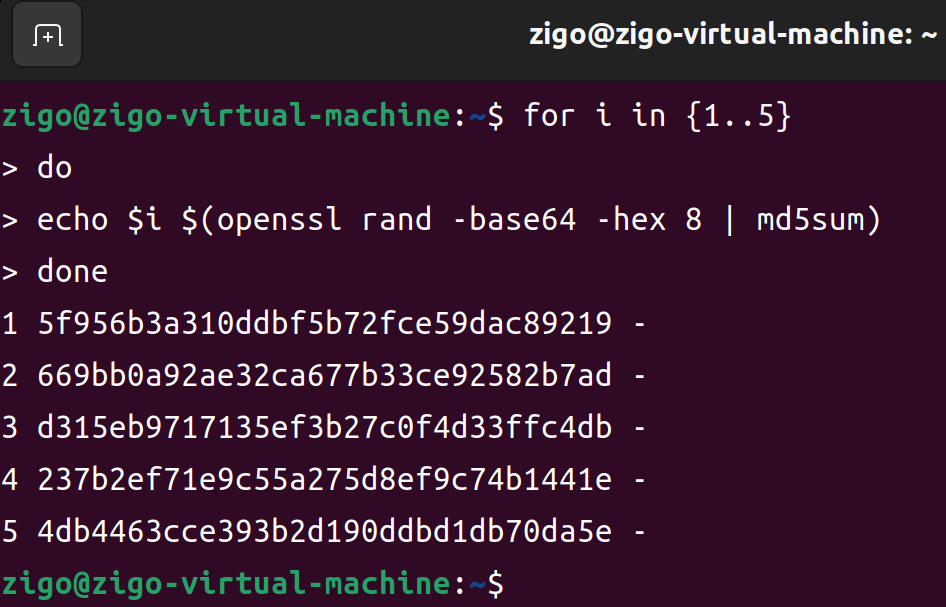
2：使用date +%N%s 通过Linux的时间戳来获取随机数。%s为获取秒级别的时间戳；%N为获取纳秒级别的时间戳。使用%s%N组合随机数，重复概率大大降低但是生成的随机数在较短时间内可能较为接近，熵来源较低。

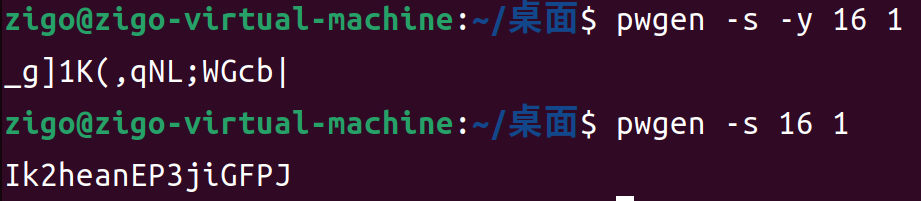


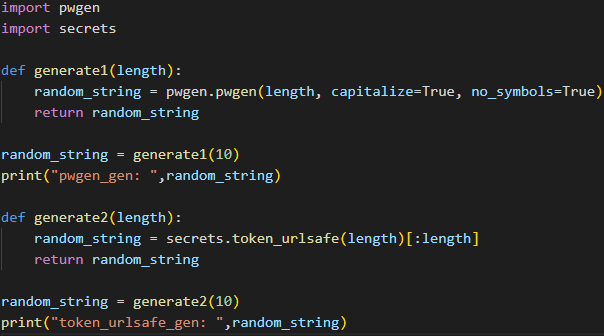
1. 调用密码程序库接口生成随机串

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。



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。





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。

1. **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





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:**  315c4eeaa8b5f8aaf9174145bf43e1784b8fa00dc71d885a804e5ee9fa40b16349c146fb778cdf2d3aff021dfff5b403b510d0d0455468aeb98622b137dae857553ccd8883a7bc37520e06e515d22c954eba5025b8cc57ee59418ce7dc6bc41556bdb36bbca3e8774301fbcaa3b83b220809560987815f65286764703de0f3d524400a19b159610b11ef3e**密文#2:**  234c02ecbbfbafa3ed18510abd11fa724fcda2018a1a8342cf064bbde548b12b07df44ba7191d9606ef4081ffde5ad46a5069d9f7f543bedb9c861bf29c7e205132eda9382b0bc2c5c4b45f919cf3a9f1cb74151f6d551f4480c82b2cb24cc5b028aa76eb7b4ab24171ab3cdadb8356f  **密文#3:**  32510ba9a7b2bba9b8005d43a304b5714cc0bb0c8a34884dd91304b8ad40b62b07df44ba6e9d8a2368e51d04e0e7b207b70b9b8261112bacb6c866a232dfe257527dc29398f5f3251a0d47e503c66e935de81230b59b7afb5f41afa8d661cb  **密文#4:**  32510ba9aab2a8a4fd06414fb517b5605cc0aa0dc91a8908c2064ba8ad5ea06a029056f47a8ad3306ef5021eafe1ac01a81197847a5c68a1b78769a37bc8f4575432c198ccb4ef63590256e305cd3a9544ee4160ead45aef520489e7da7d835402bca670bda8eb775200b8dabbba246b130f040d8ec6447e2c767f3d30ed81ea2e4c1404e1315a1010e7229be6636aaa  **密文#5:**  3f561ba9adb4b6ebec54424ba317b564418fac0dd35f8c08d31a1fe9e24fe56808c213f17c81d9607cee021dafe1e001b21ade877a5e68bea88d61b93ac5ee0d562e8e9582f5ef375f0a4ae20ed86e935de81230b59b73fb4302cd95d770c65b40aaa065f2a5e33a5a0bb5dcaba43722130f042f8ec85b7c2070  **密文#6:**  32510bfbacfbb9befd54415da243e1695ecabd58c519cd4bd2061bbde24eb76a19d84aba34d8de287be84d07e7e9a30ee714979c7e1123a8bd9822a33ecaf512472e8e8f8db3f9635c1949e640c621854eba0d79eccf52ff111284b4cc61d11902aebc66f2b2e436434eacc0aba938220b084800c2ca4e693522643573b2c4ce35050b0cf774201f0fe52ac9f26d71b6cf61a711cc229f77ace7aa88a2f19983122b11be87a59c355d25f8e4  **密文#7:**  32510bfbacfbb9befd54415da243e1695ecabd58c519cd4bd90f1fa6ea5ba47b01c909ba7696cf606ef40c04afe1ac0aa8148dd066592ded9f8774b529c7ea125d298e8883f5e9305f4b44f915cb2bd05af51373fd9b4af511039fa2d96f83414aaaf261bda2e97b170fb5cce2a53e675c154c0d9681596934777e2275b381ce2e40582afe67650b13e72287ff2270abcf73bb028932836fbdecfecee0a3b894473c1bbeb6b4913a536ce4f9b13f1efff71ea313c8661dd9a4ce  **密文#8:**  315c4eeaa8b5f8bffd11155ea506b56041c6a00c8a08854dd21a4bbde54ce56801d943ba708b8a3574f40c00fff9e00fa1439fd0654327a3bfc860b92f89ee04132ecb9298f5fd2d5e4b45e40ecc3b9d59e9417df7c95bba410e9aa2ca24c5474da2f276baa3ac325918b2daada43d6712150441c2e04f6565517f317da9d3  **密文#9:**  271946f9bbb2aeadec111841a81abc300ecaa01bd8069d5cc91005e9fe4aad6e04d513e96d99de2569bc5e50eeeca709b50a8a987f4264edb6896fb537d0a716132ddc938fb0f836480e06ed0fcd6e9759f40462f9cf57f4564186a2c1778f1543efa270bda5e933421cbe88a4a52222190f471e9bd15f652b653b7071aec59a2705081ffe72651d08f822c9ed6d76e48b63ab15d0208573a7eef027  **密文#10:**  466d06ece998b7a2fb1d464fed2ced7641ddaa3cc31c9941cf110abbf409ed39598005b3399ccfafb61d0315fca0a314be138a9f32503bedac8067f03adbf3575c3b8edc9ba7f537530541ab0f9f3cd04ff50d66f1d559ba520e89a2cb2a83  **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",

    "466d06ece998b7a2fb1d464fed2ced7641ddaa3cc31c9941cf110abbf409ed39598005b3399ccfafb61d0315fca0a314be138a9f32503bedac8067f03adbf3575c3b8edc9ba7f537530541ab0f9f3cd04ff50d66f1d559ba520e89a2cb2a83",

    "32510ba9babebbbefd001547a810e67149caee11d945cd7fc81a05e9f85aac650e9052ba6a8cd8257bf14d13e6f0a803b54fde9e77472dbff89d71b57bddef121336cb85ccb8f3315f4b52e301d16e9f52f904"

]

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)

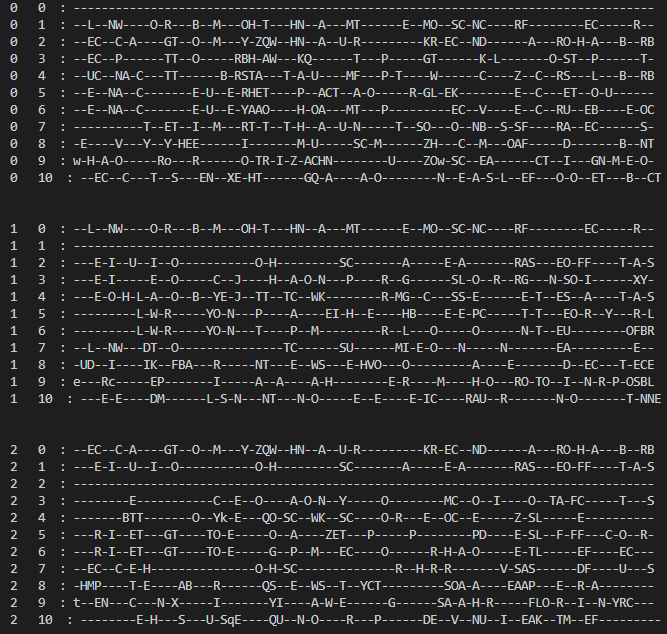
    return plaintext

xor\_result = xor(cipherText)

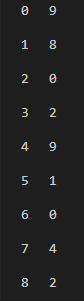
xor\_ascii\_result = hex\_to\_ascii(xor\_result)

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

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。



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。

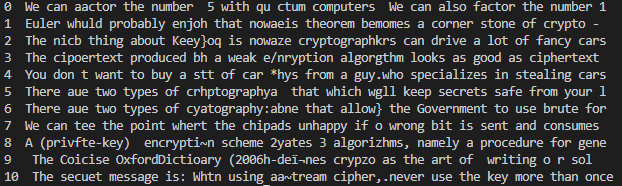
第三步，从第一位开始，将记录下来的对应编号的位置的密文与空格异或，由于已知明文异或密钥后得到密文，且明文推测为空格，因此将密文与空格异或将得到密文。将得到的所有密钥拼接起来即为所求密钥。



第四步，将密钥与密文异或解密得到明文。



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。



我们对编号10的解密结果进行人工校验：

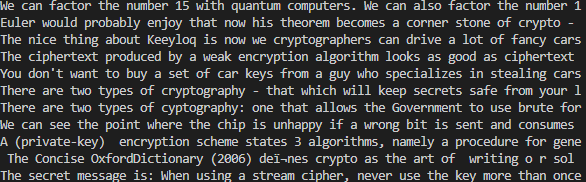
The secret message is: When using a stream cipher, never use the key more than once

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：



解密结果：



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