Crypto学习笔记

Phoenix

2021 年 3 月 12 日

目录

第1章　概述

作为一只萌新，入门CTF先学的是Crypto（手动狗头），先阐述一些心得体会。

首先，Crypto题目对数学功底要求很高，根据我目前的做题经验来说，主要难点还是在于数学和算法。其次，对选手的综合能力要求很高，一名合格的Crypto选手首先需要有较强的观察能力，能迅速识别题目中的密码类型，其次要掌握对应的攻击算法和相应的分析能力，另外就是编程水平，对自己算法的复杂度有着清晰的认知（对，说的就是上来就穷举的，doge），然后是学习能力，与时俱进，最后是广泛的知识面，能解那些跨领域结合的题目。

总的算起来我学习python+Crypto一共也就15天。(大萌新.jpg，sigh）基本上也是一边做题一边学理论，这种思路应该还是比较合理的。目前在密码学这块还有一些内容没有学习，主要是AES/DES、hash、数字签名、离散对数、椭圆曲线等。之所以打算整理这样一篇学习笔记，一是前期的时候没有对其中的理论进行完整的架构，而是许多细节，包括数学的证明、算法的剖析这一块还不够深入，正好利用几天时间重新研读。

所有的题解基本是用python。（主要是sagemath，mathematic那些还不会用，无法接受.jpg）前几章理论部分的内容基本是对一些资料的搬运与整理，题解部分主要阐述解题的思路，也希望再二次解题过程中对一些算法做一些改进。

第2章　编码

编码是将信息从一种形式或格式转换为另一种形式，解码则是将编码信息转化为原始信息，在Crypto编码是解题的基础。

2.1　hex

hex是将信息转化为16进制,密码学中大部分操作都是进行数学计算的过程。我们无法直接对字符串进行数学计算，所以需要将字符串转换为数字。可以通过hex编码的方式进行转换，将原始的字符串转化为十六进制字符串，再进行进一步的数学计算。以下为demon。

```
from Crypto.Util.number import long_to_bytes, bytes_to_long
s = 'flag'
s_by = bytes(s, 'UTF-8')  # 转换成编码
print(s_by, type(s_by))
s_str = s_by.hex()  # 转换成16进制字符串
print(str, type(str))
s_int = int(s_str, 16)  # 转换成10进制整数
print(s_int, type(s_int))
s_hex = hex(s_int)  # 转换成16进制字符串，也可以直接+0x
print(s_hex, type(s_hex))
print(ord('a'))  # 对单字符可以直接用ord函数
num = 584734024210391580014049650557280915516226103165
num_str = hex(num)
print(num_str, type(num_str))  # hex结果是字符串
```

```
num_hex = num_str[2:]# 截取第三个到最后
print(num_hex)  #若有L，则[2:-1]截取到倒数第二个字符串，前闭后开
flag = ''
for i in range(0, len(num_hex), 2):
    tmp = num_hex[i:i + 2]
    flag += chr(int(tmp, 16))
print(flag)
print(bytes_to_long(bytes("flag{this_is_a_flag}", 'UTF-8')))
print(long_to_bytes(num).decode())
'''
b'flag' <class 'bytes'>
<class 'str'> <class 'type'>
1718378855 <class 'int'>
0x666c6167 <class 'str'>
97
0x666c61677b746869735f69735f615f666c61677d <class 'str'>
666c61677b746869735f69735f615f666c61677d
flag{this_is_a_flag}
584734024210391580014049650557280915516226103165
flag{this_is_a_flag}
'''
```

2.2　urlencode

urlencode主要用于浏览器和网站之间的数据交换，这种编码是在特殊字符的hex基础上，每个字符前置一个“%”，demon如下：

```
import urllib.parse
print(urllib.parse.quote('flag{url_encode_1234_!@#$}'))
#quote处理字符转义，在特殊字符hex的基础上，每个字符前置一个%
d={'name':'bibi@flappypig,club','flag':'flag{url_encode_1234_!@#$}'}
print(urllib.parse.urlencode(d))
```

```
'''
flag%7Burl_encode_1234_%21%40%23%24%7D
name=bibi%40flappypig%2Cclub&flag=flag%7Burl_encode_1234_%21%40%23%24%7D
quote对字符串进行url编码，可以使用unquote函数进行解码
urlencode函数对字典模式的键值对进行url编码
'''
```

2.3　morsecode

摩斯电码由长音和短音构成的，使用“.”表示短音，“-”表示长音，“/”表示分隔符，解码可以用在线工具列举如下：

https://www.atool99.com/morse.php

http://www.zhongguosou.com/zonghe/moErSiCodeConverter.aspx

http://www.bejson.com/enc/morse/

https://www.jb51.net/tools/morse.htm

如果碰到摩斯电码与MISC音频题结合起来出题，求稳可以用Cool Edit进行编辑，可以明显观察长音和短音，（自信听力好可以直接听）然后将电码抄录，最后解码。用python进行解码的code如下：

```
alphabet_to_morse = {
    "A": ".-",
    "B": "-.-.",
    "C": "-.-.",
    "D": "-..",
    "E": ".",
    "F": "..-.",
    "G": "--.",
    "H": "....",
    "I": "..",
    "J": ".---",
    "K": "-.-",
    "L": ".-..",
```

```
    "M": "--",
    "N": "-.",
    "O": "---",
    "P": ".--.",
    "Q": "--.-",
    "R": ".-.",
    "S": "...",
    "T": "-",
    "U": "..-",
    "V": "...-",
    "W": ".--",
    "X": "-..-",
    "Y": "-.--",
    "Z": "--..",
    "0": "-----",
    "1": ".----",
    "2": "..---",
    "3": "...--",
    "4": "....-",
    "5": ".....",
    "6": "-....",
    "7": "--...",
    "8": "---..",
    "9": "----.",
    ".": ".-.-.-",
    ",": "--..--",
    ":": "---...",
    ";": "-.-.-.",
    "?": "..--..",
    "-": "-....-",
    "_": "..--.-",
```

```
    "(": "-.--.",
    ")": "-.--.-",
    "=": "-...-",
    "+": ".-.-.",
    "/": "-..-.",
    "@": ".--.-.",
    "$": "...-..-",
    "&": ".-...",
    "'": ".----.",
    "!": "-.-.--",
    '"': ".-..-.",
}
morse_to_alphabet = {v: k for k, v in alphabet_to_morse.items()}
def _morseremoveunusablecharacters(uncorrected_string):
    return filter(lambda char: char in alphabet_to_morse,
                  uncorrected_string.upper())
'''filter() 函数用于过滤序列，过滤掉不符合条件的元素，
返回由符合条件元素组成的新列表function -- 判断函数。iterable -- 可迭代对象
匿名函数lambda: 是指一类无需定义标识符（函数名）的函数或子程序。
lambda 函数可以接收任意多个参数 (包括可选参数) 并且返回单个表达式的值。
'''
def morseencode(decoded):
    """
    :param decoded:
    :return:
    """
    morsestring = []
    decoded = _morseremoveunusablecharacters(decoded)
    decoded = decoded.upper()
    words = decoded.split(' ')
    for word in words:
```

```
        letters = list(word)
        morseword = []
        for letter in letters:
            morseletter = alphabet_to_morse[letter]
            morseword.append(morseletter)
        word = "/".join(morseword)
        morsestring.append(word)
    return " ".join(morsestring)
def morsedecode(encoded):
    """
    :param encoded:
    :return:
    """
    characterstring = []
    words = encoded.split(" ")
    for word in words:
        letters = word.split("/")
        characterword = []
        for letter in letters:
            characterletter = morse_to_alphabet[letter]
            characterword.append(characterletter)
        word = "".join(characterword)
        characterstring.append(word)
    return " ".join(characterstring)
```

2.4　jsfuck

jsfuck仅使用6个字符“()+[]!”就可以书写任意的Javascript代码，可以在下面两个网站解码：

http://www.jsfuck.com/

https://utf-8.jp/public/jsfuck.html

2.5　uuencode

uuencode是一种将二进制文本转化为可见字符文本的一种编码，编码之后出现的是ASC码32-95的字符，也就是没有小写字母，还是比较容易识别的。也有在线解码网站：
https://www.qqxiuzi.cn/bianma/uuencode.php

2.6　base

Crypto一定要提到大名鼎鼎的base家族，如base64、base32、base16等，还有一些可能不太熟悉的，如base36、base58、base62、base85、base91、base92。其中base16就是hex，也就是用16个字符去表示256个字符（4bit的内容去表示16bit的内容），如果用更多的字符就也就是后面的base32、base64。那么如何识别base家族呢，首先看结尾有没有补位的“=”如果有就一定是，没有再看对应的字母是不是在base家族对应的字符集上。下面列举base家族对应的字符集

base16	0-9,A-F共16个以及补位的“=”
base32	A-Z,2-7共32个以及补位的“=”
base64	a-z,0-9,A-F,+,/共64个以及补位的“=”

下面举个base64例子说明它是如何将3个字符用4个字符进行表示的

A	S	T	
65	83	84	
01000001	01010011	01010100	
010000	010101	001101	010100
16	21	13	20

如上所示，先将找字符对应的asc码，再转化为二进制，然后6bit一组切分，将6bit组成的数字转化为十进制，最后查表得到字符。其它如base32也是进行类似操作。

至于没有列举的（不太常见的）可以自行查阅字符集，有些题目可能就是会用多种base进行多次编码，相应的也需要多次解码，所以判断在哪一次用了那种编码就十分重要，下面是一个非常好用的在线解码网站(对ctf来说都很好用)，能进行base家族的多种编码解码
http://ctf.ssleye.com/
最后Python demon如下

```
import base64
s = 'flag'
s_by = bytes(s, 'UTF-8')  # 转换成编码
print(s_by, type(s_by))
s_base64_by = base64.b64encode(s_by)
print(s_base64_by, type(s_base64_by))  # 类型还是编码
s_base64 = s_base64_by.decode()
print(s_base64, type(s_base64))  # decode变为字符串
print(base64.b16encode(bytes('flag', 'UTF-8')).decode())
print(base64.b32encode(bytes('flag', 'UTF-8')).decode())
print(base64.b64encode(bytes('flag', 'UTF-8')).decode())
print(base64.b16decode("666C6167".upper()))
print(base64.b32decode("MZWGCZY="))  # 结果为编码
print(base64.b64decode("ZmxhZw=="))  # 结果为编码
'''
b'flag' <class 'bytes'>
b'ZmxhZw==' <class 'bytes'>
ZmxhZw== <class 'str'>
666C6167
MZWGCZY=
ZmxhZw==
b'flag'
b'flag'
b'flag'
'''
```

第3章　古典密码

3.1　移位密码

3.1.1　简单移位密码

密码和编码最大的区别是多了一个密钥k，一般用m代表明文，c代表密文。

移位密码是将明文根据密钥进行了位置的变换而得到的密文，举个例子：

m = ''$flag\{easy\_easy\_crypto\}$''

k = ''3124''

首先以k的长度切分m如下：

$flag$　　$\{eas$　　y_ea　　sy_c　　$rypt$　　$o\}$

然后按照密钥3124的顺序对每一部分密钥进行变化：

$lafg$　　$ea\{s$　　$\_eya$　　y_sc　　$yprt$　　$\}o$

所以密文为$lafgea\{s_eyay_scyprt\}o$

再说攻击策略，一般有两种：爆破和语义分析，爆破首先爆破字段长度然后爆破顺序，具体此处不细叙。

Python加密解密代码如下

```
def shift_encrypt(m, k):
    l = len(k)
    c = ""
    for i in range(0, len(m), l):
        tmp_c = [""] * l  # 初始化列表
        if i + l > len(m):
```

```
            tmp_m = m[i:]
        else:
            tmp_m = m[i:i + l]
        for kindex in range(len(tmp_m)):
            tmp_c[int(k[kindex]) - 1] = tmp_m[kindex]
        c += "".join(tmp_c)
    return c
def shift_decrypt(c, k):
    l = len(k)
    m = ""
    for i in range(0, len(c), l):
        tmp_m = [""] * l
        if i + l > len(c):
            tmp_c = c[i:]
            use = []
            for kindex in range(len(tmp_c)):
                use.append(int(k[kindex]) - 1)
            use.sort()
            for kindex in range(len(tmp_c)):
                tmp_m[kindex] = tmp_c[use.index(int(k[kindex]) - 1)]
                # index返回第一个匹配的索引位置
        else:
            tmp_c = c[i:i + l]
            for kindex in range(len(tmp_c)):
                tmp_m[kindex] = tmp_c[int(k[kindex]) - 1]
        m += "".join(tmp_m)
    return m
m = "flag{easy_easy_crypto}"
k = "3124"
c = shift_encrypt(m, k)
print(c)
```

```
print(shift_decrypt(c, k))
# lafgea{s_eyay_scyprt}o
# flag{easy_easy_crypto}
```

3.1.2　云影密码

云影密码仅包含01248五个数字，其中0用于分割，其余数字用于做加和操作之后转换为明文，Python解码如下

```
def c01248_decode(c):
    l=c.split("0")
    origin="abcdefghijklmnopqrstuvwxyz"
    r=""
    for i in l:
        tmp=0
        for num in i:
            tmp+=int(num)
        r+=origin[tmp-1]
    return r
print(c01248_decode("884210122048022440401422420248012 2"))
# welldone
```

3.1.3　栅栏密码

栅栏密码密钥只有一个数字k，表示栅栏长度，将加密的明文分成k个一组，然后取每组的一个字符依次连接，拼接而成就是密文。样例如下：

m = "$flag\{zhalan\_mima\_hahaha\}$"

k=4

首先每4个分一组

$flag$　$\{zha$　$lan\_$　$mima$　$\_hah$　$aha\}$

然后依次取出每组的第一个字符组成一组，再依次取出第2个、第3个、第4个：

$f\{lm\_alzaihhahnmaaga\_ah\}$
而栅栏密码的解密方法就是加密的逆过程，Python加解密代码如下

```
def zhalan_encrypt(m,k):
    chip=[]
    for i in range(0,len(m),k):
        if i+k>=len(m):
            tmp_m=m[i:]
        else:
            tmp_m=m[i:i+k]
        chip.append(tmp_m)
    c=""
    for i in range(k):
        for tmp_m in chip:
            if i<len(tmp_m):
                c+=tmp_m[i]
    return c
def zhalan_decrypt(c,k):
    l=len(c)
    partnum=l//k
    if l%k!=0:
        partnum+=1
    m=[""]*l
    for i in range(0,l,partnum):
        if i + partnum>=len(c):
            tmp_c=c[i:]
        else:
            tmp_c=c[i:i+partnum]
        for j in range(len(tmp_c)):
            m[j*k+i//partnum]=tmp_c[j]
    return "".join(m)
m="flag{zhalan_mima_hahaha}"
```

```
k=4
c=zhalan_encrypt(m,k)
print(c)
print(zhalan_decrypt(c,k))
# f{lm_alzaihhahnmaaga_ah}
# flag{zhalan_mima_hahaha}
```

3.2　替代密码

单表替代密码

3.2.1　凯撒密码

凯撒密码通过将字母移动一定的位数来实现加密和解密，明文中所有的字母都在字母表上向后(或向前)按照一个固定的数目移动偏移后被替换成密文。因此，位数就是凯撒密码加密和解密的密钥因为只考虑可见字符，并且都是ASCⅡ码，所以128是模数。而只对凯撒密码的爆破就是爆破密钥K。具体见以下代码：

```
def caesar_encrypt(m, k):
    r = ""
    for i in m:
        r += chr((ord(i) + k) % 128)
    return r
def caesar_decrypt(c, k):
    r = ""
    for i in c:
        r += chr((ord(i) - k) % 128)
    return r
def caesar_brute(c, match_str):
    result = []
```

```
    for k in range(128):
        tmp = caesar_decrypt(c, k)
        if match_str in tmp:
            result.append(tmp)
    return result
m = "flag{kaisamima}"
k = 3
c = caesar_encrypt(m, k)
print(c)
print(caesar_decrypt(c, k))
c1 = "39.4H/?BA2,0.2@.?J"
print(caesar_brute(c, "flag{"))
# iodj~ndlvdplpd
# flag{kaisamima}
# ['flag{kaisamima}']
```

3.2.2　ROT13

在凯撒密码中有一种特例，当k=13，并且只用于大小写英文字母的时候，称之为ROT13.而且值得注意的是ROT13的加密和解密函数是一样的（因为英文字母只有26个，+13和-13效果一样）。

另外ROT13通常会作用在MD5，flag等字符串上，MD5中的字符只有“ABCDEF”，对应的ROT13为“NOPQRS”,flag对应的是“SYNT”，如果看到这些就可以识别出是ROT13，样例如下

```
import codecs
def rot13(m):
    r = ""
    for i in m:
        if ord(i) in range(ord('A'), ord('Z') + 1):
            r += chr((ord(i) + 13 - ord('A')) % 26 + ord('A'))
        elif ord(i) in range(ord('a'), ord('z') + 1):
            r += chr((ord(i) + 13 - ord('a')) % 26 + ord('a'))
```

```
        else:
            r += i
    return r
c = "2cf24dba5fb0a30e26e83b2ac5b9e29e1b161e5c1fa7425e73043362938b9824"
print(rot13(c))
print(rot13(rot13(c)))
print(codecs.decode(c,"rot13"))
# iodj~ndlvdplpd
# flag{kaisamima}
# ['flag{kaisamima}']
```

3.2.3　埃特巴什码

埃特巴什码的替代表不是通过移位得到的，而是通过对称获得的。其通过将字母表的位置完全镜面对称后获得字母的替代表，然后进行加密。Python代码如下

```
def atabsh_encode(m):
    alphabet="ABCDEFGHIJKLMNOPQRSTUVWXYZ"
    Origin=alphabet+alphabet.lower()
    TH_A=alphabet[::-1]
    TH_a=alphabet.lower()[::-1]
    TH=TH_A+TH_a
    r=""
    for i in m:
        tmp=Origin.find(i)
        if tmp!=-1:
            r+=TH[tmp]
        else:
            r+=i
    return r
m="flag{ok_atabsh_flag}"
c=atabsh_encode(m)
```

```
print(atabsh_encode(c))
print(c)
# uozt{lp_zgzyhs_uozt}
# flag{ok_atabsh_flag}
```

3.2.4　经典单表替代密码

经典单表替代密码就是用一个替代表对每一个位置的字符进行查表替换。因为是单表替换，所以没有替代表时，爆破的难度会比较高，一般来说会给出一段具有足够语言意义的密文，然后使用词频统计的方法进行攻击。在有替代表的情况下的加解密代码如下

```
def substitution_encode(m, k, origin="abcdefghijklmnopqrstuvwxyz"):
    r = ""
    for i in m:
        if origin.find(i) != -1:
            r += k[origin.find(i)]
        else:
            r += i
    return r
def substitution_decode(c, k, origin="abcdefghijklmnopqrstuvwxyz"):
    r = ""
    for i in c:
        if k.find(i) != -1:
            r += origin[k.find(i)]
        else:
            r += i
    return r
m = "flag{good_good_study}"
k = "zugxjitlrkywdhfbnvosepmacq"
c = substitution_encode(m, k)
print(c)
print(substitution_decode(c, k))
```

```
# iwzt{tffx_tffx_osexc}
# flag{good_good_study}
```

3.2.5　培根密码

培根密码一般使用两种不同的字体代表密文，密文的内容不是关键所在，关键是字体。使用AB代表两种字体，五个一组，表示密文，明密文对应表如下所示

a	AAAAA	g	AABBA	n	ABBAA	t	BAABA
b	AAAAB	h	AABBb	o	ABBAB	u-v	BAABB
c	AAABA	i-j	ABAAA	p	ABBBA	w	BABAA
d	AAABB	j	ABAAB	q	ABBBB	x	BABAB
e	AABAA	k	ABABA	r	BAAAA	y	BABBA
f	AABAB	l	ABABB	s	BAAAB	z	BABBB

也可以使用在线工具解密：http://rumkin.com/tools/cipher/baconian.php

3.2.6　图形密码

猪圈密码和跳舞的小人都是典型的图形替代密码，图形替代是通过将明文用图形替代以实现加密。猪圈密码使用不同的格子来表示不同的字母；跳舞的小人源自《福尔摩斯探案集》，是使用小人图案来表示不同的字母，同时用举旗子来表示单词结束。具体此处不细说，遇到可在网上查找对应表。

3.2.7　仿射密码

仿射密码替代表的生成方式依据：c=am+b mod n，其中m为明文对应字母得到的数字，n为字符数量，c为密文，a和b为密钥。在拥有密钥的情况下，解密只需要求出a关于n的逆元即可，即m=invert(a)*(c-b) mod n,关于逆元会在后面叙述，此处略过。

因为明密文空间一样，所以n很容易得知。那么在没有密钥的情况下，一般有以下几种思路：第一种是爆破，在密钥空间小的情况下可以这么做；

第二种因为仿射密码也是单表替代密码的特例，字母也是一一对应的，所以也可以使用词频统计进行攻击；第三种是已知明文攻击，如果我们知道了任意两个字符的明密文对，那么我们可以推理出密钥ab。具体实现如下

```
from gmpy2 import invert
def affine_encode(m, a, b, origin="abcdefghijklmnopqrstuvwxyz"):
    r = ""
    for i in m:
        if origin.find(i) != -1:
            r += origin[(a * origin.index(i) + b) % len(origin)]
        else:
            r += i
    return r
def affine_decode(c, a, b, origin="abcdefghijklmnopqrstuvwxyz"):
    r = ""
    n = len(origin)
    ai = invert(a, n)
    for i in c:
        if origin.find(i) != -1:
            r += origin[(ai * (origin.index(i) - b)) % len(origin)]
        else:
            r += i
    return r
def affine_guessab(m1, c1, m2, c2, origin="abcdefghijklmnopqrstuvwxyz"):
    x1 = origin.index(m1)
    x2 = origin.index(m2)
    y1 = origin.index(c1)
    y2 = origin.index(c2)
    n = len(origin)
    dxi = invert(abs(x1 - x2), n)
    if x1 - x2 < 0:
        dxi *= -1
```

```
    a = dxi * (y1 - y2) % n
    b = (y1 - a * x1) % n
    return (a, b)
m = "affinecipher"
a = 5
b = 8
c = affine_encode(m, a, b)
print(c)
print(affine_decode(c, a, b))
print(affine_guessab('a', 'i', 'f', 'h'))
# ihhwvcswfrcp
# affinecipher
# (mpz(5), mpz(8))
```

多表替代密码

3.2.8　棋盘密码

Playfair、Polybius和Nihilist均属于棋盘类密码。此类密码的密钥为1个5×5的棋盘。棋盘的生成符合如下条件：顺序随意；不得出现重复字母；i和j可视为同一字（也有讲q去除的，以保证总数为25个）。生成棋盘后，不同的加密方式使用了不同的转换方式。生成棋盘代码如下

```
def gen_cheese_map(k, use_Q=True, upper=True):
    k = k.upper()
    k0 = ""
    origin = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
    for i in k:
        if i not in k0:
            k0 += i
    for i in origin:
        if i not in k0:
```

```
            k0 += i
    if use_Q == True:
        k0 = k0[0:k0.index("J")] + k0[k0.index("J") + 1:]
    else:
        k0 = k0[0:k0.index("Q")] + k0[k0.index("Q") + 1:]
    if upper == False:
        k0 = k0.lower()
    assert len(k0) == 25
    r = []
    for i in range(5):
        r.append(k0[i * 5:i * 5 + 5])
    return r
print(gen_cheese_map("helloword"))
# ['HELOW', 'RDABC', 'FGIKM', 'NPQST', 'UVXYZ']
```

Playfair根据明文的位置去寻找新的字母。首先将明文字母两两一组进行切分，并按照如下规则进行加密。

1）若两个字母不同行也不同列，则需要在矩阵中找出另外两个字母（第一个字母对应行优先），使这四个字母成为一个长方形的四个角。

2）若两个字母同行，则取这两个字母右方的字母（若字母在最右方则取最左方的字母）。

2）若两个字母同列，则取这两个字母下方的字母（若字母在最下方则取最上方的字母）。

变换、加密和解密代码如下

```
def _playfair_2char(tmp, map):
    for i in range(5):
        for j in range(5):
            if map[i][j] == tmp[0]:
                ai = i
                aj = j
            if map[i][j] == tmp[1]:
                bi = i
```

```
                bj = j
    if ai == bi:
        axi = ai
        bxi = bi
        axj = (aj + 1) % 5
        bxj = (bj + 1) % 5
    elif aj == bj:
        axj = aj
        bxj = bj
        axi = (ai + 1) % 5
        bxi = (bi + 1) % 5
    else:
        axi = ai
        axj = bj
        bxi = bi
        bxj = aj
    return map[axi][axj] + map[bxi][bxj]
def playfair_encode(m, k="", cheese_map=[]):
    m = m.upper()
    origin = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
    tmp = ""
    for i in m:
        if i in origin:
            tmp += i
    m = tmp
    assert k != "" or cheese_map != []
    if cheese_map == []:
        map = gen_cheese_map(k)
    else:
        map = cheese_map
    m0 = []
```

```
    idx = 0
    while idx < len(m):
        tmp = m[idx:idx + 2]
        if tmp[0] != tmp[1]:
            m0.append(tmp)
            idx += 2
        elif tmp[0] != "X":
            m0.append(tmp[0] + "X")
            idx += 1
        else:
            m0.append(tmp[0] + "Q")
            idx += 1
        if idx == len(m) - 1:
            if tmp[0] != "X":
                m0.append(tmp[0] + "X")
                idx += 1
            else:
                m0.append(tmp[0] + "Q")
    r = []
    for i in m0:
        r.append(_playfair_2char(i, map))
    return r
def _playfair_2char_decode(tmp, map):
    for i in range(5):
        for j in range(5):
            if map[i][j] == tmp[0]:
                ai = i
                aj = j
            if map[i][j] == tmp[1]:
                bi = i
                bj = j
```

```
    if ai == bi:
        axi = ai
        bxi = bi
        axj = (aj - 1) % 5
        bxj = (bj - 1) % 5
    elif aj == bj:
        axj = aj
        bxj = bj
        axi = (ai - 1) % 5
        bxi = (bi - 1) % 5
    else:
        axi = ai
        axj = bj
        bxi = bi
        bxj = aj
    return map[axi][axj] + map[bxi][bxj]
def playfair_decode(c, k="", cheese_map=[]):
    assert k != "" or cheese_map != []
    if cheese_map == []:
        map = gen_cheese_map(k)
    else:
        map = cheese_map
    r = []
    for i in c:
        r.append(_playfair_2char_decode(i, map))
    return "".join(r)
m = "Hide the gold in the tree stump"
k = "playfairexample"
c = playfair_encode(m, k)
print(c)
print(playfair_decode(c, k))
```

```
# ['BM', 'OD', 'ZB', 'XD', 'NA', 'BE', 'KU', 'DM', 'UI', 'XM', 'MO', 'UV', 'IF'
# HIDETHEGOLDINTHETREXESTUMP
```

Polybius密码相对简单一点（Nihilist原理相同），只用棋盘的坐标作为密文，可能不一定是数字，坐标可以用字母表示，解密也是参照map去寻找对应的明文，代码如下

```
def polybius_encode(m, k="", name="ADFGX", cheese_map=[]):
    m = m.upper()
    assert k != "" or cheese_map != []
    if cheese_map == []:
        map = gen_cheese_map(k)
    else:
        map = cheese_map
    r = []
    for x in m:
        for i in range(5):
            for j in range(5):
                if map[i][j] == x:
                    r.append(name[i] + name[j])
    return r
def polybius_decode(c, k="", name="ADFGX", cheese_map=[]):
    assert k != "" or cheese_map != []
    if cheese_map == []:
        map = gen_cheese_map(k)
    else:
        map = cheese_map
    r = ""
    for x in c:
        i = name.index(x[0])
        j = name.index(x[1])
        r += map[i][j]
    return r
```

```
m = "helloworld"
k = "abcd"
c = polybius_encode(m, k)
print(c)
print(polybius_decode(c, k))
# ['DF', 'AX', 'FA', 'FA', 'FG', 'XD', 'FG', 'GD', 'FA', 'AG']
# HELLOWORLD
```

3.2.9　维吉尼亚密码

凯撒密码是单表替代密码，其只使用了一个替代表，维吉尼亚密码则是标准的多表替代密码。

首先，多表替代密码的密钥不再是固定不变的，而是随着位置发生改变的。在维吉尼亚密码中，根据密钥的字母来选择。比如密钥是LOVE，那么明文会没四组一个循环，使用的密钥如表所示

L-LMNOPQRSTUVWXYZABCDEFGHIJK
O-OPQRSTUVWXYZABCDEFGHIJKLMN
V-VWXYZABCDEFGHIJKLMNOPQRSTU
E-EFGHIJKLMNOPQRSTUVWXYZABCD

明文的第一个位置用“L”加密，二三四用“OVE”，到第五个位置回到“L”，一般情况下，维吉尼亚密码的破解必须依靠爆破+词频统计的方法来进行，这里有一个破解网站：http://www.quipqiup.com/index.php（具体使用参见网站说明）

对于自己破解维吉尼亚密码来说，其实也是词频统计，因为间隔密钥长度地字符使用的替代表是相同的。所以，如果密文长度足够长，并且知道密钥长度，是可以通过词频统计的方法进行破解的。

关于密钥的长度，可以使用卡西斯基试验和弗里德曼试验来获取，卡西斯基试验是类似于the这样的常用单词，如果使用了重复的密钥加密，那么两个相同的连续串的间隔将是密钥长度的倍数。获取这样的值并计算最大公约数，就可以得到密钥长度。

3.2.10　希尔密码

希尔密码是运用基本矩阵原理的替换密码，由Lester S.Hill在1929年发明。将每个字母当作二十六进制数字：A=0，B=1，C=2,依次类推。将一串字母当成你、维向量，与一个n×n的矩阵相乘，再将得出的结果模26。注意，用作加密的矩阵（即密钥）在Z_26^n中必须是可逆的，否则就不可能译码。只有矩阵的行列式与26互质才是可逆的。例如明文act：

$$\begin{bmatrix} 0 \\ 2 \\ 19 \end{bmatrix}$$

密钥：

$$\begin{bmatrix} 6 & 24 & 1 \\ 13 & 16 & 10 \\ 20 & 17 & 15 \end{bmatrix}$$

加密过程为：

$$\begin{bmatrix} 6 & 24 & 1 \\ 13 & 16 & 10 \\ 20 & 17 & 15 \end{bmatrix} \begin{bmatrix} 0 \\ 2 \\ 19 \end{bmatrix} \equiv \begin{bmatrix} 67 \\ 222 \\ 319 \end{bmatrix} \equiv \begin{bmatrix} 15 \\ 14 \\ 7 \end{bmatrix} \mod 26$$

所以密文为POH

第4章　RSA

4.1　数学知识预备

4.1.1　模逆元

若ab≡1 (mod n) 则称a和b关于模n互为模逆元。求模逆元的算法有多种，这里主要用扩展Euclidean算法，首先描述Euclidea算法，它可以给出两个正整数a和b的最大公因子。Euclidean算法首先令r_0为a，令r_1为b，然后执行如下除法运算：

$$\begin{aligned} r_0 &= q_1 r_1 + r_2 \qquad 0 < r_2 < r_1 \\ r_1 &= q_2 r_2 + r_3 \qquad 0 < r_3 < r_2 \\ &\cdots\cdots \\ r_{m-2} &= q_{m-1} r_{m-1} + r_m \qquad 0 < r_m < r_{m-1} \\ r_{m-1} &= q_m r_m \end{aligned}$$

Euclidean算法的伪代码描述如下给出

在该算法中容易看到

$$gcd(r_0, r_1) = gcd(r_1, r_2) = \ldots = gcd(r_{m-1}, r_m) = r_m$$

因此，可以得出$gcd(a, b) = r_m$。

由于Euclidean算法能计算出最大公因子，它可以用来判断一个正整数b¡n是否有模逆元，通过调用Euclidean(n,b)来检查一下是否有r_m=1.但是，这并没有计算出模逆元的值（如果其存在）。

现在，假定按照下面构造定义了两个数列（其中q_j按照Euclidean算法定义）：

$$t_0, t_1, \ldots, t_m \qquad s_0, s_1, \ldots, s_m$$

```
1: function EUCLIDEAN(a, b)
2:     r_0 ← a
3:     r_1 ← b
4:     m ← 1
5:     while r_m ≠ 0 do
6:         q_m ← ⌊(r_{m-1} − 1)/r_m⌋
7:         r_{m+1} ← r_{m-1} − q_m r_m
8:         m ← m + 1
9:     end while
10:    m ← m − 1
11:    return (q_1, ..., q_m; r_m)
12:    Comment: r_m = gcd(a, b)
13: end function
```

其中

$$t_j = \begin{cases} 0 & j = 0 \\ 1 & j = 1 \\ t_{j-2} - q_{j-1}s_{j-1} & j \geq 2 \end{cases} \quad s_j = \begin{cases} 1 & j = 0 \\ 0 & j = 1 \\ s_{j-2} - q_{j-1}s_{j-1} & j \geq 2 \end{cases}$$

那么，有如下结果对于$0 \leq j \leq m$,有$r_j = s_j r + 0 + t_j r_1$

证明：对j用归纳法证明。对于j=0和j=1命题是平凡的。假定命题对于j=i-1和j=i-2成立，其中i≥2；下证对j=i-2成立。由归纳假定，有

$$r_{i-2} = s_{i-2}r_0 + t_{i-2}r_1 \qquad r_{i-1} = s_{i-1}r_0 + t_{i-1}r_1$$

现在计算

$$\begin{aligned} r_i &= r_{i-2} - q_{i-1}r_{i-1} \\ &= s_{i-2}r_0 + t_{i-2}r_1 - q_{i-1}(s_{i-1}r_0 + t_{i-1}r_1) \\ &= (s_{i-2} - q_{i-1}s_{i-1})r_0 + (t_{i-2} - q_{i-1}t_{i-1})r_1 \\ &= s_i r_0 + t_i r_1 \end{aligned}$$

因此，由归纳法，命题对于所有j≤0成立。

接下来给出扩展Euclidean算法，它以两个整数a和b作为输入，计算出整

数r，s和t使得r=gcd(a,b)且sa+tb=r。在这个版本中，不必记录所有的q_j, r_j, s_j, t_j;在算法的任意一点上，记录每个数列的最后两项就够了。
下面推论可由之前结论直接推出：
假定gcd(r_0, r_1)=1。那么$r_1^{-1} mod r_0 = t_m mod r_0$。
证明，由之前结论，有

$$1 = gcd(r_0, r_1) = s_m r_0 + t_m r_1$$

两边模r_0约化等式，得到

$$t_m r_1 \equiv 1 (mod r_0)$$

即得所证。
下面举个例子，计算$28^{-1} mod 75$,有如下计算

i	r_j	q_j	s_j	t_j
0	75		1	0
1	28	2	0	1
2	19	1	1	-2
3	9	2	-1	3
4	1	9	3	-8

因此我们发现3×75-8×25=1，应用推论可得到$28^{-1} mod 75 = -8 mod 75 = 67$
扩展Euclidean算法立即得出b^{-1}mod a(如果存在)。事实上，模逆元b^{-1}mod a=t mod a；这可由刚才推论直接导出。然而，一个更有效的算法是把所有的s_j的计算都从算法中去掉，并在主循环中每次度模a约化t，得到下面的模逆元算法。

1: **function** EXTENDED EUCLIDEAN(a, b)
2: $a_0 \leftarrow a$
3: $b_0 \leftarrow b$
4: $t_0 \leftarrow 0$
5: $t \leftarrow 1$
6: $s_0 \leftarrow 1$
7: $s \leftarrow 0$
8: $q \leftarrow \lfloor \frac{a_0}{b_0} \rfloor$
9: $r \leftarrow a_0 - qb_0$
10: **while** $r > 0$ **do**
11: $temp \leftarrow t_0 - qt$
12: $t_0 \leftarrow t$
13: $t \leftarrow temp$
14: $temp \leftarrow s_0 - qs$
15: $s_0 \leftarrow s$
16: $s \leftarrow temp$
17: $a_0 \leftarrow b_0$
18: $b_0 \leftarrow r$
19: $q \leftarrow \lfloor \frac{a_0}{b_0} \rfloor$
20: $r \leftarrow a_0 - qb_0$
21: **end while**
22: $r \leftarrow b_0$
23: **return** (r, s, t)
24: Comment: $r = gcd(a, b) and sa + tb = r$
25: **end function**

```
1: function MULTIPLICATIVE INVERSE(a, b)
2:     a_0 ← a
3:     b_0 ← b
4:     t_0 ← 0
5:     t ← 1
6:     q ← ⌊a_0 / b_0⌋
7:     r ← a_0 − q b_0
8:     while r > 0 do
9:         temp ← (t_0 − qt) mod a
10:        t_0 ← t
11:        t ← temp
12:        a_0 ← b_0
13:        b_0 ← r
14:        q ← ⌊a_0 / b_0⌋
15:        r ← a_0 − q b_0
16:    end while
17:    if b_0 ≠ 1 then
18:        b has no inverse modulo a
19:    else
20:        return t
21:    end if
22: end function
```

4.1.2 中国剩余定理

假定$m_1,\ldots,m_n$为两两互素的正整数，又假定$a_1,\ldots,a_n$为整数，那么同余方程组 x≡ $a_i(modm_i)(1\le i\le n)$有模M=$m_1\times\ldots\times m_n$的唯一解

$$x=\sum_{i=1}^{n}a_iM_it_imodM$$

其中$M_i=M/m_i$,且$t_i=M_i^{-1}modm_i$，$1\le i\le n$。
证明：由假设可知，gcd(m_i,m_j)=1，所以gcd(m_i,M_i)=1，这说明M_i对m_i存在模逆元，记为t_i。考察乘积$a_iM_it_i$可知：

$$a_iM_it_i\equiv a_i*1\equiv a_i(modm_i)$$

对$j\neq i$，有$a_iM_it_i\equiv 0(modm_j)$,所以$x=\sum_{i=1}^{n}a_iM_it_i\equiv a_i+0\equiv a_i(modm_i)$

4.2 RSA密码体制

Alice向Bob广播了一句信息：“Bob老师，我有重要情报汇报！”

Bob很聪明，他知道Alice发现了什么但是不能通过广播的方式直接汇报，于是它立即生成两个大素数p和q，通过乘法计算出了n=p*q，并取了一个合适的素数e，并且向所有人包括Alice传播了信息(n,e)，也就是加密密钥，当然攻击者Cat也通过广播通信截取到了(n,e)。

Alice拿到(n,e)之后，首先将重要信息通过hex和padding转换成一串数字m，然后进行一次计算得到了c：c=m^emod n，并通过广播通信将c传给Bob，此时攻击者Cat也监听到了c。

Bob比Cat多掌握了p和q的值，Bob通过计算e关于n的欧拉函数的逆元，可以求出d，即满足e*d=1 mod $\varphi(n)$,$\varphi(n)$是n的欧拉函数，又因为n=p*q，所以$\varphi(n)$=(p-1)*(q-1)。所以Bob又多掌握了一个d，通过d进行如下计算 m=c^dmod n。

至此，Bob通过hex处理，掌握了Alice的情报。而对于Cat来说，由于它们没有掌握p和q的值，所以无法计算出d从而无法解密c得到m

(d,n)即为次密码的私钥。

以上就是RSA加密解密过程。

4.3 素性检测

在建立RSA密码体制的过程中，首先要生成大的“随机素数”，具体方法则是生成大的随机整数，然后检验素性。目前已经有多项式时间的确定性算法，因此RSA体制是可行的。

再提一个有用的事实，根据素数个数定理，定义$\pi(N)$为小于等于N的素数个数，$\pi(N)$约等于N/lnN。因此一个随机的512比特整数为素数的概率大约为$1/ln2^{512}$约等于1/355，如果是奇数则加倍。所以事实上能有效生成“很可能是素数”的大随机整数。

4.3.1 Legendre和Jacobi符号

定义：假定p为一个奇素数，a为一个整数。a定义为模p的二次剩余，如果a≠0(mod p)，且同余方程$y^2 \equiv a(modp)$有一个解y。否则a定义为模p的二次非剩余，如果a≠0(mod p)，且a不是模p的二次剩余。

下面证明一个结果：Euler准则，可以为二次剩余问题提供一个多项式时间的确定性算法

Euler准则：设p为一个奇素数，a为一个正整数，那么a是一个模p的二次剩余，当且仅当

$$a^{(p-1)/2} \equiv 1(modp)$$

证明：首先，假定$y^2 \equiv a(modp)$，有费马小定理可知，如果p是素数，那么$a^{p-1} \equiv 1(modp)$对于任意a≠0(mod p)成立。于是有

$$\begin{aligned} a^{(p-1)/2} &\equiv (y^2)^{(p-1)/2}(modp) \\ &\equiv y^{p-1}(modp) \\ &\equiv 1(modp) \end{aligned}$$

反过来，假定$a^{(p-1)/2} \equiv 1(modp)$。设b为一个模p的本原元素。那么$\equiv b^i(modp)$对于某个正整数i成立，有

$$\begin{aligned} a^{(p-1)/2} &\equiv (b^i)^{(p-1)/2}(modp) \\ &\equiv b^{i(p-1)/2}(modp) \end{aligned}$$

由于p的阶为p-1，因此必有p-1整除i(p-1)/2。因此i是偶数，于是a的平方根为$\pm b^{i/2} mod p$。

利用模p指数的平方-乘算法，上面给出了判定二次剩余问题的多项式时间算法，时间复杂度为O($(logp)^3$)。

下面给出更多定义。

假定p是一个奇素数。对任何整数a，定义Legendre符号$\left(\frac{a}{p}\right)$如下：

$$\left(\frac{a}{p}\right) = \begin{cases} 0 & a \equiv 0(modp) \\ 1 & a\ is\ quardratic\ residue \\ -1 & a\ is\ quardratic\ non-residue \end{cases}$$

结合欧拉准则，容易有如下结果
假定p是一个奇素数。那么

$$\left(\frac{a}{p}\right) \equiv a^{(p-1)/2}(modp)$$

接下来，定义Legendre符号的一般形式。
假定n是一个奇正整数，且n的素数幂因子分解为

$$n = \prod_{i=1}^{k} p_i^{e_i}$$

设a为一个整数。那么Jacobi符号$\left(\frac{a}{n}\right)$定义为：

$$\left(\frac{a}{n}\right) = \prod_{i=1}^{k} \left(\frac{a}{p_i}\right)^{e_i}$$

4.3.2 Solovay-Strassen算法和Miller-Rabin算法

下面给出两种判定素数的方法，关于它们的细节和正确性就不赘述。可以证明它们都时多项式时间算法，在单次时，前者具有1/2的错误概率，后者至多有1/4的错误概率，并且在n为素数的情况下不会回答n为合数。错误概率会随次数增加呈指数递减。它们的时间复杂度都为O($(logn)^3$)，但实际运行中，后者比前者要好。

```
1: function SOLOVAY-STRASSEN(n)
2:     choose a random integer a(1 ≤ a ≤ n − 1)
3:     x ← (a/n)
4:     if x = 0 then
5:         return "n is composite"
6:     end if
7:     y ← a^((p−1)/2)(modn)
8:     if x ≡ y(modn) then
9:         return "n is prime"
10:    else
11:        return "n is composite"
12:    end if
13: end function
```

```
1: function SOLOVAY-STRASSEN(n)
2:     n − 1 = 2^k m,m is odd
3:     choose a random integer a(1 ≤ a ≤ n − 1)
4:     b ← a^m modn
5:     if b ≡ 1(modn) then
6:         return "n is prime"
7:     end if
8:     for i ← 0 to k-1 do
9:         if b ≡ −1(modn) then
10:            return "n is prime"
11:        else
12:            b ← b^2 modn
13:        end if
14:    end for
15:    return "n is composite"
16: end function
```

4.4 分解因子算法

攻击RSA最明显的方式就是试图分解公开模数。对于大整数最有效的三种算法是二次筛法、椭圆曲线分解算法和数域筛法。其它作为先驱的著名算法包括Pollard的ρ方法和p-1算法、William的p+1算法、连分式算法，当然还有最简单的试除法。下面就简单介绍一种。

4.4.1 Pollard p-1算法

```
1: function POLLARD P-1 FACTORING(n, B)
2:     b ← 2
3:     for j ← 2 to B do
4:         a ← a^j mod n
5:     end for
6:     d ← gcd(a − 1, n)
7:     if 1 < d < n then
8:         return d
9:     else
10:        return "faliure"
11:    end if
12: end function
```

在Pollard p-1算法中：B是预先指定的一个界，假定p是n的一个素因子，又假定对每一个素数幂$q|(p-1)$，有q≤B。那么在这种情形下必有

$$(p-1)|B!$$

在for循环结束时，我们有

$$a \equiv 2^{B!}(mod\ n)$$

由于$p|n$，一定有

$$a \equiv 2^{B!}(mod\ p)$$

又由Fermat定理可知

$$2^{p-1} \equiv 1(mod\ p)$$

由于$(p-1)|B!$，于是有

$$a \equiv 1(modp)$$

因此有$p|(a-1)$。由于我们已经有p—n，可以看到p—d，其中d=gcd(a-1,n)。整数d就是n的一个非平凡因子(除非a=1)。一旦找到n的一个非平凡因子d，就可以对d和n/d继续分解(如果d和n/d还是合数)。

4.5 CTF Crypto中RSA的常见题型

4.5.1 直接模数分解

破解RSA的难点主要在于分解n，先前也提到了一些分解的算法，如果题中的n是可以直接分解的，一般用以下两种方法，一是用神器yafu分解，二是一个比较好的分解网站http://factordb.com/index.php。若以上两个都无法分解n，那么这题肯定是用其它方法。下面给出标准的RSA解密过程。

```
from Crypto.Util.number import long_to_bytes
from gmpy2 import modinv
n = 413769611561689481963033842189411173678500\
195092020490260371681949822197841 59
e = 65537
c = 416129310053087483683642260362520682458969\
393304043278907405416527408727 2587
# from yafu
p = 130451115685568383270871808144053983073
q = 317183651045971246384245233153006206783
# calc
d = modinv(e, (p - 1) * (q - 1))
m = pow(c, d, n)
print(long_to_bytes(m).decode())
# flag(rsa_256bit_brute)
```

4.5.2 公约模数分解

如果Alice和Bob之间进行了两次消息传递过程，且Bob不小心在两次通信中生成的大素数有一个是相同的，那么可以通过对两次通信的n求公约数进而分解两次通信的n。

4.5.3 其它模数分解

有的时候题目中会给出一些其它信息来辅助分解n。举个例子，有一题在提供了(n,e,c)的同时，还给出了npp的值，npp的值为(p+1)*(q+2)。那么构造一个方程:$x^2-(p+q)x+p*q=0$，此方程中(p+q)的值和pq的值都可以通过npp和n转化得到，而此方程的两个根即为p和q的值。

4.5.4 小指数明文爆破

如果Bob使用的e太小了，比如e=3，且Alice要传给Bob的明文也很小，比如就几个字节，那么由c=$m^e mod m$，e=3，m很小，n很大，所以可能发生$m^e < n$，那么此时c=m^3，直接对c开三次方根即可得到m。当然这是一种极端的情况。如果$m^3 > n$但是没有超过太多，即$k*n < m^3 < (k+1)*n$，且k是可以爆破大小的，则可以通过关系式k*n+c=m^3来爆破明文。

4.5.5 LLL-attack

如果Bob使用的e是3，同时Alice习惯性地在密文前面附上了一句众所周知的问候语，也就是说明，密文的一部分已经泄露，或者Bob不小心泄露了生成的p或者q的一部分，又或者Bob泄露了明文的部分bit。那么在泄露的信息长度足够的时候，可以通过Coppersmith method的方法求得明文。

假设已知m的一部分为base，此时可以使用LLLattack的方法进行攻击。目前GitHub上提供给了进行LLLattack的sage代码，地址为
https://github.com/mimoo/RSA-and-LLL-attacks。

如遇到这类题，可以使用sage-online解决，修改LLLattack的源代码进行攻击。以上方法同样适用p泄露了三分之二的情况。给出的代码在泄漏的字节略小于三分之二时也是可以接受的，在条件合适的情况下，1024bit的p只

泄露了576bit的情况下也可以成功破解，链接地址为
http://inaz2.hatenablog.com/entries/2016/01/20。

4.5.6 Wiener Attack & Boneh Durfee Attack

上两节的情况都是Bob选取的e太小造成的。如果Bob选择的e很大，但产生的d太小，也会被成功攻击。这就是RSA Wiener Attack。

一般来说，如果e很大，远远超过了65537，那么基本确定就是Wiener Attack。在GitHub上也有成功攻击的脚本，地址为
https://github.com/pablocelayes/rsa-wiener-attack。

4.5.7 共模攻击

Bob为了省事，在两次通信中使用了相同的n，而Alice是对相同的m加密。这时，Cat可以不计算d而直接计算m的值。

想要使用共模攻击的前提是有两组及以上的RSA加密过程，而且其中的m和n都是相同的。Python代码如下

```
from gmpy2 import invert
def same_n_attack(n, e1, e2, c1, c2):
    def egcd(a, b):
        x, lastx = 0, 1
        y, lasty = 1, 0
        while (b != 0):
            q = a // b
            a, b = b, a % b
            x, lastx = lastx - q * x, x
            y, lasty = lasty - q * y, y
        return (lastx, lasty)
    s = egcd(e1, e2)
    s1 = s[0]
    s2 = s[1]
    if s1 < 0:
```

```
        s1 = -s1
        c1 = invert(c1, n)
    elif s2 < 0:
        s2 = -s2
        c2 = invert(c2, n)
    m = (pow(c1, s1, n) * pow(c2, s2, n)) % n
    return m
```

4.5.8 广播攻击

Bob不仅多次选择的加密指数低（例如3或其它较小的素数），而且这几次加密过程，加密的信息都是相同的，那么由：

$$c_1 \equiv m^e mod n_1$$

$$c_2 \equiv m^e mod n_2$$

$$c_3 \equiv m^e mod n_3$$

那么通过中国剩余定理，可以计算出一个数c_x为：

$$c_x \equiv m^3 mod n_1 n_2 n_3$$

然后对c_x开三次方即可得到m。Python代码如下

```
from gmpy2 import iroot
def broadcast_attack(data):
    def extended_gcd(a,b):
        x, y = 0, 1
        lastx, lasty = 1, 0
        while b:
            a,(q,b)=b,divmod(a,b)
            x, lastx = lastx - q * x, x
            y, lasty = lasty - q * y, y
        return (lastx, lasty,a)
    def chinese_remainder_theorem(items):
```

```
        N = 1
        for a, n in items:
            N *= n
        result = 0
        for a, n in items:
            m = N // n
            r,s,d=extended_gcd(n,m)
            if d!=1:
                N=N//n
                continue
            result += a * s * m
        return result % N
    x = chinese_remainder_theorem(data)
    m = iroot(x, 3)[0]
    return m
# 其中data=[(c1,n1),(c2,n2),(c3,n3)]
```

4.5.9 相关消息攻击

当Alice使用同一对公钥对两个具有某种线性关系的消息M_1和M_2进行加密，并将加密后的消息C_1、C_2发送给Bob时，我们可以获得对应的消息M_1与M_2、这里假设模数为N，两者之间的线性关系为$M_1 = a * M_2 + b$，那么当e=3时可以得到：

$$M_2 = \frac{2a^3bC^2 - b^4 + C_1b}{aC_1 - a^4C_2 + 2ab^3} = \frac{bC_1 + 2a^3C_2 - b^3}{aC_1 - a^3C_2 + 2b^3}$$

```
from gmpy2 import invert
def relate_message_attack(a, b, c1, c2, n):
    b3 = pow(b, 3, n)
    part1 = b * (c1 + 2 * c2 - b3) % n
    part2 = a * (c1 - c2 + 2 * b3) % n
    part2 = invert(part2, n)
    return part1 * part2 % n
```

第5章　CryptoHack题解

先附上平台网址：https://cryptohack.org/

5.1　INTRODUCTION

CHALLENGES

5.1.1　Finding Flags

题目大意是每题都要提交正确的flag才能通过，本题的flag已经在题目中直接给出。复制粘贴提交即可。

5.1.2　Great Snakes

本题下载运行附带的Python脚本即可得到flag。

5.1.3　Network Attacks

本题下载示例脚本，大致发现是通过telnetlib库对socket.cryptohack.org 11112端口以JSON对象的方式进行请求，将clothes修改为flag即可拿到flag。另外，本题的示例脚本在以后与服务器的交互中依然有用，可以保存，以备修改。

5.2 GENERAL

ENCODING

5.2.1 ASCII

本题将转化为数字转化为ASC码即可。

```
lst=[99, 114, 121, 112, 116, 111, 123, 65, 83, 67, 73,
73, 95, 112, 114, 49, 110, 116, 52, 98, 108, 51, 125]
for i in lst:
    print(chr(i),end='')
```

5.2.2 Hex

本题是将十六进制字符串解码，方法多样，以下列举三种，可以使用库函数，也可以手动解码。

```
from binascii import a2b_hex,unhexlify
source='63727970746f7b596f755f77696c6c5f62655f776f726b696e' \
       '675f776974685f6865785f737472696e67735f615f6c6f747d'
flag = ''
for i in range(0, len(source), 2):
    tmp = source[i:i + 2]
    flag += chr(int(tmp, 16))
print(flag)
print(a2b_hex(source).decode())
print(unhexlify(source).decode())
```

5.2.3 Base64

题意是将十六进制字符串解码为字节，然后将其编码为Base64。

```
import base64
s = "72bca9b68fc16ac7beeb8f849dca1d8a783e8acf9679bf9269f7bf"
s_by=base64.b16decode(s.upper())
s_base64=base64.b64encode(s_by)
print(s_base64.decode())
```

5.2.4 Bytes and Big Integers

本题要求将大整数转化为消息，并提示了库函数，直接用即可。

```
from Crypto.Util.number import long_to_bytes
num= #以后过长的数据就不显示了，具体见题目
print(long_to_bytes(num).decode())
```

5.2.5 Encoding Challenge

结合题目和13377的源码分析，本题要求使用脚本自动化解密100次来拿到flag。加密类型有base64,hex,rot13,bigint,utf-8,用我们先前所学过的对应地解密即可。成功解密会返回下一次的加密信息，知道100次，中间不能出错，一旦出错就会终止。

```
import telnetlib
import base64
import json
from Crypto.Util.number import long_to_bytes
import codecs
from binascii import unhexlify
HOST = "socket.cryptohack.org"
PORT = 13377
tn = telnetlib.Telnet(HOST, PORT)
def readline():
    return tn.read_until(b"\n")
def json_recv():
```

```
    line = readline()
    return json.loads(line.decode())
def json_send(hsh):
    request = json.dumps(hsh).encode()
    tn.write(request)
challenge_words = []  # 建立接收数据的列表
for i in range(100):
    received = json_recv()
    if received["type"] == "base64":  # 根据编码类型解码加入到列表中
        challenge_words.append(base64.b64decode
        (received["encoded"].encode()).decode())
    elif received["type"] == "hex":
        challenge_words.append(unhexlify(received["encoded"]).decode())
    elif received["type"] == "rot13":
        challenge_words.append(codecs.decode(received["encoded"], 'rot_13'))
    elif received["type"] == "bigint":
        challenge_words.append(long_to_bytes
        (int(received["encoded"], 16)).decode())
    elif received["type"] == "utf-8":
        challenge_words.append(''.join(chr(b) for b in received["encoded"]))
    print("%d : %s" % (i, challenge_words[i]))  # 打印出本次要发送到服务端的值
    to_send = {"decoded": challenge_words[i]}
    json_send(to_send)
received = json_recv()  # 接收flag数据
print(received["flag"])
```

XOR

5.2.6 XOR Starter

将给定字符串与13异或即可。

```
s = "label"
flag = ""
for i in s:
    flag += chr(ord(i) ^ 13)
print(flag)
```

5.2.7 XOR Properties

根据异或性质求解。详见代码注释

```
from binascii import a2b_hex
from pwn import xor  # xor得到bytes,参数可以是bytes或str
KEY1 = "a6c8b6733c9b22de7bc0253266a3867df55acde8635e19c73313"
KEY2_KEY3 = "c1545756687e7573db23aa1c3452a098b71a7fbf0fdddde5fc1"
flag_123 = "04ee9855208a2cd59091d04767ae47963170d1660df7f56f5faf"
flag = xor(a2b_hex(flag_123), a2b_hex(KEY1), a2b_hex(KEY2_KEY3))
#flag = (flag ^ key1 ^ ke2 ^ key3) ^ key1 ^ (key2 ^ key3)
print(flag.decode())
```

5.2.8 Favourite byte

题意是将flag与一个字节异或，通过遍历ASC码字符拿到flag。

```
from binascii import a2b_hex
from pwn import xor
from Crypto.Util.number import long_to_bytes
s = "73626960647f6b206821204f21254f7d694f7624662065622127234f726927756d"
s1 = str(long_to_bytes(int(s, 16)))[2:-1]
for i in range(128):
    flag = xor(a2b_hex(s), i).decode()
    if "crypto{" in flag:
        print(flag, i)
```

5.2.9 You either know, XOR you don't

本题应该是与一串字节异或，提示flag格式有助于我们找到key，因此尝试将密文与crypto按照字节顺序进行异或以期得到密钥。

```
import binascii
from pwn import xor
s = "0e0b213f26041e480b26217f27342e175d0……"
string = "crypto{"
s_by = binascii.a2b_hex(s)
key = ""
for i in range(7):
    key += chr(s_by[i] ^ ord(string[i]))
print(key)  # 根据打印结果盲猜key为myXORkey
key += "y"
flag = xor(s_by, key)
print(flag.decode())
```

5.2.10 Lemur XOR

题目说用相同的密钥通过异或隐藏了两个图像，根据异或性质容易得到，将现在的两张图片按照像素点进行异或，就是原始两张图片的异或。

```
from PIL import Image  # 使用PIL的Image处理图像
im1 = Image.open('flag.png')  # 打开图片
pim1 = im1.load()
# 获取图片通道数:img.mode RGB（三通道）或者L（单通道灰度）
im2 = Image.open('lemur.png')
pim2 = im2.load()
width, height = im1.size  # 获取图片尺寸
for i in range(width):
    for j in range(height):
        r1, g1, b1 = pim1[i, j]
        r2, g2, b2 = pim2[i, j]
```

```
            pim1[i, j] = (r1 ^ r2, g1 ^ g2, b1 ^ b2)
im1.show()  # 显示图片
```

MATHEMATICS

5.2.11 Greatest Common Divisor

求最大公约数，题目描述已经很详细，用Euclidean algorithm就行，之前也提过。顺便提及，本模块的几道题都可以用库函数（摊手）。

```
def gcd(a, b):
    while b != 0:
        r = a % b
        a = b
        b = r
    return a
a, b = 66528, 52920
print(gcd(a, b))
```

5.2.12 Extended GCD

扩展欧几里得算法，前文也提及，不赘述。

```
def extended_gcd(a, b):
    if (b == 0):
        x, y = 1, 0
        return (x, y)
    x, y = extended_gcd(b, a % b)
    t = x
    x = y
    y = t - a // b * y
    return (x, y)
a, b = 26513, 32321
print(extended_gcd(a, b))
```

5.2.13 Modular Arithmetic 1

求模数，简洁明了。

```
a1, b1 = 11, 6
a2, b2 = 8146798528947, 17
print(min(a1 % b1, a2 % b2))
```

5.2.14 Modular Arithmetic 2

幂次方求模，建议用快速幂，库函数中的pow函数和以下代码性能相差无几。

```
def pow(a, p, k):  # 快速幂a^p%k
    tmp = 1
    a %= k
    while p != 0:
        if p % 2 == 1:
            tmp = tmp * a % k
        a = a * a % k
        p = p // 2
    return tmp % k
print(pow(273246787654,65536,65537))
```

5.2.15 Modular Inverting

求模逆元，前文有提，以下有尾递归之嫌，并不好，可有更好实现，或用库函数。

```
def egcd(a, b):
    if a == 0:
        return (b, 0, 1)
    else:
        g, y, x = egcd(b % a, a)
        return (g, x - (b // a) * y, y)
```

```
def modinv(a, m):
    g, x, y = egcd(a, m)
    if g != 1:
        raise Exception('modular inverse does not exist')
    else:
        return x % m
print(modinv(3,13))
```

DATA FORMATS

5.2.16　Privacy-Enhanced Mail?

本题介绍了PEM格式，要求提取公钥。这里需要使用openssl，命令行使用openssl rsa -in private_rsa.pem -text -out private.txt(文件名自行修改)命令进行打印信息，得到密钥转10进制。

```
s='''
7c:3b:1d:53:4f:29:9b:43:c1:26:08:76:30:3c:0a:
95:be:17:bf:91:a5:df:2f:1c:ac:da:7c:75:a0:23:
6e:4f:81:e1:21:0d:27:c0:12:6f:b3:4d:80:f2:7a:
41:a4:d7:e4:8c:a7:c5:b0:e7:88:78:b1:9f:d0:d6:
c0:bf:68:30:fb:8a:44:01:b1:6d:93:8a:d5:4c:4d:
0b:35:68:62:05:6c:b0:55:4e:b2:ab:83:90:ad:18:
25:b3:1d:af:bf:2f:c0:5d:19:4f:38:c2:f2:24:20:
d3:21:0a:da:02:30:24:26:40:ca:e0:05:eb:85:cb:
c8:dc:ca:18:25:ea:74:96:d9:b1:70:c5:cb:fe:35:
4f:e1:9a:63:10:2b:82:f3:8d:5d:7c:25:17:35:20:
8b:83:a5:42:40:92:7f:89:98:48:c1:6a:5f:e7:0c:
e9:50:da:ff:7b:f9:f4:b7:1b:59:81:01:a5:20:48:
cd:30:c1:6c:b9:94:33:0b:10:59:2d:2c:95:d4:d0:
e5:79:f5:28:7f:f7:4a:88:26:8d:03:89:69:8c:8f:
7b:9a:e8:13:f3:92:46:89:3d:02:66:1c:f0:8d:9c:
```

```
bc:ec:9f:72:2c:f7:6d:0e:96:f1:e1:77:37:e2:9e:
ce:86:76:76:7c:b6:e1:df:0d:bd:2d:73:1e:d8:48:
b1'''
lst=s.split("\n")
s="".join(lst)
lst=s.split(":")
flag="".join(lst)
print(int(flag,16))
```

5.2.17　CERTainly not

与上题类似，用openssl x509 -inform der -in certificate.der -out certificate.pem进行格式转化再输出。另外提一下用openssl x509 -in mycert.pem -text -noout是打印到终端(控制不输出任何密钥信息)

```
s='''
b4:cf:d1:5e:33:29:ec:0b:cf:ae:76:f5:fe:2d:
c8:99:c6:78:79:b9:18:f8:0b:d4:ba:b4:d7:9e:02:
52:06:09:f4:18:93:4c:d4:70:d1:42:a0:29:13:92:
73:50:77:f6:04:89:ac:03:2c:d6:f1:06:ab:ad:6c:
c0:d9:d5:a6:ab:ca:cd:5a:d2:56:26:51:e5:4b:08:
8a:af:cc:19:0f:25:34:90:b0:2a:29:41:0f:55:f1:
6b:93:db:9d:b3:cc:dc:ec:eb:c7:55:18:d7:42:25:
de:49:35:14:32:92:9c:1e:c6:69:e3:3c:fb:f4:9a:
f8:fb:8b:c5:e0:1b:7e:fd:4f:25:ba:3f:e5:96:57:
9a:24:79:49:17:27:d7:89:4b:6a:2e:0d:87:51:d9:
23:3d:06:85:56:f8:58:31:0e:ee:81:99:78:68:cd:
6e:44:7e:c9:da:8c:5a:7b:1c:bf:24:40:29:48:d1:
03:9c:ef:dc:ae:2a:5d:f8:f7:6a:c7:e9:bc:c5:b0:
59:f6:95:fc:16:cb:d8:9c:ed:c3:fc:12:90:93:78:
5a:75:b4:56:83:fa:fc:41:84:f6:64:79:34:35:1c:
ac:7a:85:0e:73:78:72:01:e7:24:89:25:9e:da:7f:
65:bc:af:87:93:19:8c:db:75:15:b6:e0:30:c7:08:
```

```
f8:59'''#去掉了开头的00
lst=s.split("\n")
s="".join(lst)
lst=s.split(":")
flag="".join(lst)
print(int(flag,16))
```

5.2.18 Transparency

题中给了PEM格式的RSA公钥。要求我们在其TLS证书中找到使用这些参数的cryptohack.org的子域，然后访问该子域以获取flag。这里可能需要去相关的网站搜索，略过不表。

5.3 MATHEMATICS

MODULAR MATH

5.3.1 Quadratic Residues

接下来几题和二次剩余有关，前文已介绍，相关算法也给出，题目描述也很详细，故直接上code。

```
def judge_qr(a, p):
    tmp = 1
    for i in range((p - 1) // 2):
        tmp = tmp * a % p
    if tmp == 1:
        return True
    else:
        return False
p = 29
ints = [14, 6, 11]
```

```
for i in ints:
    print(judge_qr(i, p))
for i in range(100):
    if i * i % p == 6:
        print(i)
```

5.3.2 Legendre Symbol

```
# p和ints见output
def judge_qr(a, p):
    if pow(a, (p - 1) // 2, p) == 1:
        return True
    else:
        return False
for i in ints:
    if judge_qr(i, p):
        ans = i
        print(ans)
print(pow(ans,(p+1)//4,p))
'''
p=4*k-1
a^((p-1)/2)=1 mod p 即 a^(2*k-1)-1 | p
即a*a^(2*k-1)-1=a^2k-a | p
a^2k = a mod p (a^k)^2=a mod p
'''
```

5.3.3 Modular Square Root

```
def judge_qr(a, p):
    if pow(a, (p - 1) // 2, p) == 1:
        return True
    else:
```

```
        return False
def Tonel(a, p):
    q = p - 1
    s = 0
    z = 0
    while q % 2 == 0:
        q = q // 2
        s += 1
    if s == 1:
        return pow(a, (p + 1) // 4, p)
    for i in range(1, p):
        if judge_qr(i, p) == False:
            z = i
            break
    c = pow(z, q, p)
    r = pow(a, (q + 1) // 2, p)
    t = pow(a, q, p)
    m = s
    while t % p != 1:
        i0 = 0
        for i in range(1, m):
            if pow(t, int(math.pow(2, i)), p) == 1:
                i0 = i
                break
        b = pow(c, int(math.pow(2, m - i0 - 1)), p)
        r = r * b % p
        t = t * b * b % p
        c = b * b % p
        m = i0
    return r
# a,p见output
```

```
print(Tonel(a, p))
```

5.3.4 Chinese Remainder Theorem

由于后续在RSA题目中还会碰见中国剩余定理，故本题并没有写一个通用的函数。

```
from gmpy2 import iroot
a1, a2, a3 = 2, 3, 5
m1, m2, m3 = 5, 11, 17
M = m1 * m2 * m3
M1 = M // m1
M2 = M // m2
M3 = M // m3
t1 = invert(M1, m1)
t2 = invert(M2, m2)
t3 = invert(M3, m3)
x = (a1 * t1 * M1 + a2 * t2 * M2 + a3 * t3 * M3) % M
print(x % 935)
```

LATTICES

5.3.5 Vectors

5.3.6 Size and Basis

以上两题与向量的一些基本概念有关，不赘述。

5.3.7 Gram Schmidt

这题描述了施密特正交算法(未单位化)，按题目实现即可，后面还会用到。（下面代码并未给出一般化的函数，具体见后面LLL部分）另外，虽然提示要求单位化，但实际ac的flag确是未单位化的(雾)。

```
import math
class v:
    def __init__(self, a1, a2, a3, a4):
        self.a1 = a1
        self.a2 = a2
        self.a3 = a3
        self.a4 = a4
def orthonormal(u):
    t = math.sqrt(mul(u, u))
    a1 = u.a1 / t
    a2 = u.a2 / t
    a3 = u.a3 / t
    a4 = u.a4 / t
    r = v(round(a1, 7), round(a2, 7), round(a3, 7), round(a4, 7))
    return r
def sub(a, b):
    a1 = a.a1 - b.a1
    a2 = a.a2 - b.a2
    a3 = a.a3 - b.a3
    a4 = a.a4 - b.a4
    u = v(round(a1, 7), round(a2, 7), round(a3, 7), round(a4, 7))
    return u
def mul(a, b):
    r = a.a1 * b.a1 + a.a2 * b.a2 + a.a3 * b.a3 + a.a4 * b.a4
    return round(r, 7)
def multiple(k, a):
    a1 = k * a.a1
    a2 = k * a.a2
    a3 = k * a.a3
    a4 = k * a.a4
    u = v(round(a1, 7), round(a2, 7), round(a3, 7), round(a4, 7))
```

```
    return u
v1 = v(4, 1, 3, -1)
v2 = v(2, 1, -3, 4)
v3 = v(1, 0, -2, 7)
v4 = v(6, 2, 9, -5)
lst_v = []
lst_v.append(v1)
lst_v.append(v2)
lst_v.append(v3)
lst_v.append(v4)
lst_u = []
lst_u.append(v1)
for i in range(1, 4):
    u = lst_v[i]
    for j in range(0, i):
        u = sub(u, multiple(mul(lst_v[i], lst_u[j]) /
         mul(lst_u[j], lst_u[j]), lst_u[j]))
    lst_u.append(u)
tmp = []
for i in range(4):
    tmp.append(orthonormal(lst_u[i]))
    print(lst_u[i].a1,lst_u[i].a2,lst_u[i].a3,lst_u[i].a4)
    print(tmp[i].a1, tmp[i].a2, tmp[i].a3, tmp[i].a4)
```

5.3.8 What's a Lattice?

计算涉及矩阵，不赘述。

5.3.9 Gaussian Reduction

高斯约减求解二维最短向量问题。算法已经给出，注意其中(b)这一步是向下取整。

```
from math import floor
class vec:
    def __init__(self, x, y):
        self.x = x
        self.y = y
def mul(u, v):
    r = u.x * v.x + u.y * v.y
    return r
def multiple(k, a):
    x = k * a.x
    y = k * a.y
    u = vec(x, y)
    return u
def sub(u, v):
    x = u.x - v.x
    y = u.y - v.y
    r = vec(x, y)
    return r
def Gaussia(v, u):
    while True:
        if mul(u, u) < mul(v, v):
            t = u
            u = v
            v = t
        m = floor(mul(v, u) / mul(v, v))
        if m == 0:
            break
        else:
            u = sub(u, multiple(m, v))
    return (v, u)
v = vec(846835985, 9834798552)
```

```
u = vec(87502093, 123094980)
v, u = Gaussia(v, u)
print(mul(v, u))
```

5.3.10 Find the Lattice

接下来两题与格密码有关，作为萌新，只能依样画葫芦，还原算法，具体细节和证明留待日后学习。参考书籍《格理论与密码学》，网上有一篇博客也基本还原了书中的一小部分内容，地址如下：
https://blog.csdn.net/qq_33458986/article/details/104366177。

```
from Crypto.Util.number import long_to_bytes
from math import floor
class vec:
    def __init__(self, x, y):
        self.x = x
        self.y = y
def mul(u, v):
    r = u.x * v.x + u.y * v.y
    return r
def multiple(k, a):
    x = k * a.x
    y = k * a.y
    u = vec(x, y)
    return u
def sub(u, v):
    x = u.x - v.x
    y = u.y - v.y
    r = vec(x, y)
    return r
def Gaussia(v, u):
    while True:
        if mul(u, u) < mul(v, v):
```

```
            t = u
            u = v
            v = t
        m = floor(mul(v, u) / mul(v, v))
        if m == 0:
            break
        else:
            u = sub(u, multiple(m, v))
    return (v, u)
q, h = ()
e = #见output
v = vec(1, h)
u = vec(0, q)
v, u = Gaussia(v, u)
f = v.x
g = v.y
print(long_to_bytes(decrypt(q, h, f, g, e)).decode())
# f,g是基于高斯算法找到的两个基向量中绝对值小的那个
```

5.3.11 Backpack Cryptography

本题需要利用LLL算法，完全破解是272维，根据书中所说，300维以内还是比较弱的，可破解。但是我根据书中的LLL算法在python中运行性能也不佳，运行一次需要10min，而且在处理浮点数时会出现一些问题，可能是导致最终没有运行出结果的原因。（我的算法在运行书中给出的低维较小的数时是没有问题的）

```
def sub(u, v):
    r = [0]
    for i in range(1, n + 1):
        r.append(u[i]- v[i])
    return r
def mul(u, v):
```

```
    r = 0
    for i in range(1, n + 1):
        r += u[i] * v[i]
    return r
def multi(k, v):
    r = [0]
    for i in range(1, n + 1):
        r.append(k * v[i])
    return r
def orthogonal(v, k):
    u = [0]
    u.append(v[1])
    for i in range(2, k + 1):
        u.append(v[i])
        for j in range(1, i):
            u_ij = mul(v[i], u[j]) / mul(u[j], u[j])
            u[i] = sub(u[i], multi(u_ij, u[j]))
    return u
def lll(v):
    k = 2
    while k <= n:
        print(k)
        u = orthogonal(v, k)
        for j in range(1, k):
            tmp=round(mul(v[k], u[j]) / mul(u[j], u[j]))
            v[k] = sub(v[k], multi(tmp, v[j]))
        u_k=mul(v[k], u[k - 1]) / mul(u[k - 1], u[k - 1])
        if mul(u[k], u[k])/mul(u[k - 1], u[k - 1])
        >= (3 / 4 - u_k * u_k):
            k += 1
        else:
```

```
            t = v[k]
            v[k] = v[k - 1]
            v[k - 1] = t
            k = max(k - 1, 2)
    r = [0]
    for i in range(1, n + 1):
         t = []
         for j in range(n + 1):
              t.append(v[i][j])
         r.append(t)
    return r
def LLL(v):
    a = lll(v)
    b = lll(a)
    while a != b:
        a = b
        b = lll(b)
    return b
S=
o=[0,] #后面见output，注意下标从1开始
n = 272+ 1
v = [0]
for i in range(1, n):
    t = [0]
    for j in range(1, n):
        if i == j:
            t.append(2)
        else:
            t.append(0)
    t.append(o[i])
    v.append(t)
```

```
t = [0]
for i in range(1, n):
    t.append(1)
t.append(S)
v.append(t)
```

后来借助sagemath约1min能跑出正确结果，下面是后续的python处理代码。需要注意的是得到的最短向量可能需要取反（1变-1，-1变1），因为是绝对值最小，两种情况都要试一试，同时注意题目的加密应该是倒着来的，要把消息串反过来。

```
from Crypto.Util.number import long_to_bytes
o = '-1  1 -1 -1 -1 etc'#from sagemath, 直接复制
t = o.split()# 以空格为分隔符，包含 \n
s = ''
for i in range(len(t)):
    if t[i] == '-1':
        s += '1'
    else:
        s += '0'
print(long_to_bytes(int(s[::-1], 2)))
```

PROBABILITY

5.3.12　Jack's Birthday Hash

5.3.13　Jack's Birthday Confusion

这两题都是计算概率，需要稍微接触一些hash的知识，有一题的概率与生日悖论有关，日后再细究。

BRAINTEASERS PART 1

5.3.14 Successive Powers

题目中的这一串数字数某个整数的连续高次幂模三位素数，三位素数通过观察可以直接打表，后续直接根据模的性质去枚举即可。

```
prime = [853, 857, 859, 863, 877, 881, 883, 887, 907, 911,
919, 929, 937, 941, 947, 953, 967, 971, 977, 983, 991, 997]
m = [588, 665, 216, 113, 642, 4, 836, 114, 851, 492, 819, 237]
flag = True
for p in prime:
    for x in range(2, 1000):
        for i in range(len(m) - 1):
            if m[i] * x % p == m[i + 1]:
                flag = True
            else:
                flag = False
                break
        if flag:
            print(p, x)
            break
    if flag:
        break
```

5.3.15 Adrien's Signs

从source.py中不难发现，是将flag转化为二进制消息串，然后对每一位进行操作时，产生随机数e，计算n=$a^e mod p$，该位为1，则将n放入encrypt_flag中，否则将-n mod p放入其中。所以这是一道离散对数题，我们只要去计算能否得到对应的指数来复原对应的二进制消息串，进而得到flag。以下代码的性能并不佳，后续可以尝试改进。

```
from Crypto.Util.number import long_to_bytes
# encrypt_flag见output
```

```
a = 288260533169915
p = 1007621497415251
def meetMidle(p, g, h):
    # B=2^25
    B = pow(2, 25)
    x0, x1 = 0, 0
    x = -1
    # 建立字典记录
    hashMap = {}
    # 等式关系: h*g^(x1)=(g^B)^x0
    # 首先计算等式左边，将x1和对应求的值分别加入字典中
    left = h
    # g_pre=1
    for i in range(B):
        hashMap[left] = i
        # left=h/g_pre%p
        # g_pre=g_pre*g
        left = left * g % p
    # 开始遍历等式右边的内容，如果相等，则找到对应的解
    # 计算右边的底数baseRight=g^B
    baseRight = 1
    for i in range(B):
        baseRight = baseRight * g % p
    # 开始带入x0=0,1,2,3....判断(g^B)^x0是否在哈希表中
    right = 1
    for i in range(1, B + 1):
        right = right * baseRight % p
        if (right in hashMap):
            x0 = i
            x1 = hashMap[right]
            break
```

```
    x = B * x0 - x1
    if (x == -1):
        print("There is no solution")
    else:
        return x
s = ''
for i in range(56, 216): #crypto{}已知，求中间的内容
    t = meetMidle(p, a, encrypt_flag[i])
    if t == 0:
        s += '0'
    else:
        s += '1'
print(long_to_bytes(int(s, 2)))
```

5.3.16 Modular Binomials

本题类似RSA共模攻击求解二项式方程组。由题有

$$c_1 \equiv (2p+3q)^{e_1} \ mod \ N$$
$$c_2 \equiv (5p+7q)^{e_2} \ mod \ N$$

即

$$c_1^{e_2} \equiv (2p+3q)^{e_1e_2} \ mod \ N$$
$$c_2^{e_1} \equiv (5p+7q)^{e_1e_2} \ mod \ N$$

也即

$$c_1^{e_2} \equiv (2p)^{e_1e_2} + (3q)^{e_1e_2} \ mod \ N$$
$$c_2^{e_1} \equiv (5p)^{e_1e_2} + (7q)^{e_1e_2} \ mod \ N$$

令a=$2^{e_1e_2}$ mod n，b=$3^{e_1e_2}$ mod n，c=$5^{e_1e_2}$ mod n，d=$7^{e_1e_2}$ mod n，则有

$$c_1^{e_2} \equiv ap^{e_1e_2} + bq^{e_1e_2} \ mod \ N \qquad ①$$
$$c_2^{e_1} \equiv cp^{e_1e_2} + dq^{e_1e_2} \ mod \ N \qquad ②$$

①*d-②*b有

$$(ad-bc)p^{e_1e_2} \equiv dc_1^{e_2} - bc_2^{e_1} \ mod \ N$$

解得q=gcd($(ad-bc)^{-1}*(dc_1^{e_2}-bc_2^{e_1})$,N)，-1表示模逆元。q同理，当然也可以直接用N去除。

```
from gmpy2 import invert, gcd
# N,e1,e2,c1,c2见data
a = pow(2, e1 * e2, N)
b = pow(3, e1 * e2, N)
c = pow(5, e1 * e2, N)
d = pow(7, e1 * e2, N)
c1_e2 = pow(c1, e2, N)
c2_e1 = pow(c2, e1, N)
t1 = (d * c1_e2 - b * c2_e1) * invert(a * d - b * c, N)
p = gcd(t1, N)
t2 = c1_e2 * c - c2_e1 * a * invert(b * c - a * d, N)
q = gcd(t2, N)
print('crypto{' + str(p) + ',' + str(q) + '}')
```

5.3.17　Broken RSA

题目中说mess up，具体是个什么情况呢？原来是n是个素数，e是个合数(16)，虽然我们能够得到phi=n-1，但是gcd(e,phi)=8，直接就血压飙升。无奈，只能自行推导运算式。

由于有

$$m^{16} \equiv c \ mod \ n$$

$$gcd(phi//8, 8) = 1$$

不妨先考虑

$$r \equiv m^2 \ mod \ n$$

则有

$$r^8 \equiv m^{16} \equiv c \ mod \ n$$

下面考虑模逆元t，使得

$$8 * t \equiv 1 \ mod \ phi//8$$

即8*t=k*(phi//8)+1,则

$$c^t \equiv (c^{\frac{k*(phi//8)+1}{8}})^8 \equiv c^{k*phi//8+1} \equiv c\ mod\ n$$

容易知道其中运用了费马小定理$c^{phi} \equiv 1 mod n$,不难发现

$$c^t \equiv r\ mod\ n$$

接下来只需要求二次剩余即可。事实上这题的数据到这里就结束了，如果没有解出来，还有后续类似小系数爆破的操作，具体见code。这题解法像暴力开次方，似乎借助sagemath有更好操作，本萌新就先不深入探讨。

```
from Crypto.Util.number import long_to_bytes
from sympy.ntheory.residue_ntheory import sqrt_mod
import gmpy2
# n,e,c见题目文件
phi = n - 1
root_8th_of_c = pow(c, gmpy2.invert(8, phi // 8), n)
print(long_to_bytes(sqrt_mod(root_8th_of_c, n, True)[0]).decode())
# 事实上这里已经解出来了
root_8th_of_1_all = set(pow(i, (phi // 8), n) for i in range(1, 20))
root_8th_of_1_all = set(r for r in set(root_8th_of_1_all) if pow(r, 8, n) == 1)
root_8th_of_c_all = [root_8th_of_c * r % n for r in root_8th_of_1_all]
m_all = [m for r in root_8th_of_c_all for m in sqrt_mod(r, n, True)]
print(len(m_all))
for m in m_all:
    if b'crypto{' in long_to_bytes(m):
        print(long_to_bytes(m).decode())
```

BRAINTEASERS PART 2

5.3.18　Unencryptable

题目中的n是1024位，显然是unencryptable(doge)，但是注意到output有broken!?，结合加密文件分析，有一个DATA和一个判定明文和密文是否相等的条件，

在加密文件中将N替换为output中的N去加密DATA，就会发生RSA broken!?，我们不难得到$data^{65537} \equiv data \ mod \ N$，也就是$N|(data^{65536}-1)$，因此猜测$data^{65536}-1$中可能有因子p或q，接下来就是因式分解求gcd来得到N的一个因子。

```
from gmpy2 import gcd, invert
from Crypto.Util.number import long_to_bytes
# N,e,c见output,m见source
or i in range(1, 16):
    if gcd(m ** (2 ** i) - 1, N) != 1:
        p = gcd(m ** (2 ** i) - 1, N)
        break
    if gcd(m ** (2 ** i) + 1, N) != 1:
        p = gcd(m ** (2 ** i) + 1, N)
        break
q = N // p
d = invert(e, (p - 1) * (q - 1))
m = pow(c, d, N)
print(long_to_bytes(m))
```

PRIMES

5.3.19 Prime and Prejudice

总结起来这道题就是希望能找到一个绕过米勒拉宾素性检测的合数。（实际上是要根据题目名字去找到那篇paper），出去某些原因考虑，复现算法的代码就不放了。paper地址是https://eprint.iacr.org/2018/749.pdf

5.4 RSA

STARTER

5.4.1　RSA Starter 1

水题

```
print(pow(101, 17, 22663))
```

5.4.2　RSA Starter 2

水题+1

```
m, p, q, e = 12, 17, 23, 65537
n = p * q
print(pow(m, e, n))
```

5.4.3　RSA Starter 3

水题++

```
p = 857504083339712752489993810777
q = 1029224947942998075080348647219
print((p - 1) * (q - 1))
```

5.4.4　RSA Starter 4

++水题

```
from gmpy2 import invert
p = 857504083339712752489993810777
q = 1029224947942998075080348647219
print(invert(e, (p - 1) * (q - 1)))
```

5.4.5　RSA Starter 5

d见上题。

```
d = 12183288670241573157707396295737778019551049996539846984328 1
N = 88256459553622414063962598765941602942623923080461461327916 3
c = 77578995801157823671636298847186723593814843845525223303932
print(pow(c, d, N))
```

5.4.6 RSA Starter 6

数据签名，题目已经讲得比较详细。

```
import hashlib
# N,d见private.key
m = "crypto{Immut4ble_m3ssag1ng}"
H = hashlib.sha256(m.encode()).hexdigest()#H是str
S = pow(int(H,16), d, N)
print(hex(S))
```

PRIMES PART 1

5.4.7 Factoring

丢进yafu或在线网站

5.4.8 Inferius Prime

看e=3以为是小指数明文爆破，但是不成功，丢网站上居然分解成功，这就是Inferius Prime吗？（doge，代码省略）

5.4.9 Monoprime

这回n就是素数，道理一样，不是吗？

```
from gmpy2 import invert
from Crypto.Util.number import long_to_bytes
# ct,d,n见output
```

```
d=invert(e,n-1)
print(long_to_bytes(pow(ct,d,n)).decode())
```

5.4.10 Square Eyes

n是完全平方数，根据欧拉函数性质即可。

```
from gmpy2 import iroot, invert
from Crypto.Util.number import long_to_bytes
# N,c,e见output
p = iroot(N, 2)[0]
d = invert(e, p * (p - 1))
print(long_to_bytes(pow(c, d, N)).decode())
```

5.4.11 Manyprime

```
from gmpy2 import invert
from Crypto.Util.number import long_to_bytes
# n,c,e见output，p为n分解的素数列表，丢网站分解即可
tmp = 1
for i in P:
    tmp = tmp * (i - 1)
d = invert(e, tmp)
print(long_to_bytes(pow(ct, d, n)).decode())
```

PUBLIC EXPONENT

5.4.12 Salty

e=1没想到吧?(doge)

```
# ct见output
print(long_to_bytes(ct).decode())
```

5.4.13 Modulus Inutilis

本题才是小指数明文爆破。

```
from gmpy2 import iroot
from Crypto.Util.number import long_to_bytes
i = 0
while 1:
    if (iroot(ct + i * n, 3)[1] == True):
        print(long_to_bytes(iroot(ct + i * n, 3)[0]))
        print(iroot(ct + i * n, 3)[0])
        break
    i = i + 1
```

5.4.14 Everything is Big

Wiener Attack，可套用github脚本。代码过长不列。

5.4.15 Crossed Wires

多次加密就多次解密

```
from gmpy2 import invert
from Crypto.Util.number import long_to_bytes
# 相关参数见output
q = my_key[0] // p
phi = (p - 1) * (q - 1)
for i in range(4, -1, -1):
    d = invert(friend_keys[i][1], phi)
    c = pow(c, d, friend_keys[i][0])
print(long_to_bytes(c).decode())
```

5.4.16 Everything is Still Big

Boneh Durfee Attack，比Wiener Attack强一些，可套用github脚本。代码过长不列。

5.4.17 Endless Emails

广播攻击，猜测有几组加密信息是一样的，枚举即可。

```
from gmpy2 import iroot
from Crypto.Util.number import long_to_bytes
# 相关参数见output
def broadcast_attack(data):
    def extended_gcd(a, b):
        x, y = 0, 1
        lastx, lasty = 1, 0
        while b:
            a, (q, b) = b, divmod(a, b)
            x, lastx = lastx - q * x, x
            y, lasty = lasty - q * y, y
        return (lastx, lasty, a)
    def chinese_remainder_theorem(items):
        N = 1
        for a, n in items:
            N *= n
        result = 0
        for a, n in items:
            m = N // n
            r, s, d = extended_gcd(n, m)
            if d != 1:
                N = N // n
                continue
            result += a * s * m
```

```
        return result % N

    x = chinese_remainder_theorem(data)
    m = iroot(x, e)[0]
    return m
n = [n0, n1, n2, n3, n4, n5, n6]
c = [c0, c1, c2, c3, c4, c5, c6]
for i in range(5):
    for j in range(i + 1, 6):
        for k in range(j + 1, 7):
            data = [(c[i], n[i]), (c[j], n[j]), (c[k], n[k])]
            m = long_to_bytes(broadcast_attack(data))
            if b'crypto{' in m:
                print(i, j, k, )
                print(m.decode())
```

PRIMES PART 2

5.4.18 Infinite Descent

用descent.py中的递降法找到的两个素数会比较接近，所以容易(费马)分解。

5.4.19 Marin's Secrets

p，q满足$2^t - 1$的形式，直接枚举即可。

```
# n,c,e见output
from gmpy2 import invert, gcd
from Crypto.Util.number import long_to_bytes
prime = 2
for _ in range(10000):
```

```
    prime = prime << 1
    if gcd(n, prime - 1) == prime - 1:
        break
p = prime - 1
q = n // p
d = invert(e, (p - 1) * (q - 1))
m = pow(c, d, n)
print(long_to_bytes(m).decode())
```

5.4.20 Fast Primes

本题中的两个素数事实上还是比较相近的，所以可以分解（应该有更合理的办法）

5.4.21 Ron was Wrong, Whit is Right

excerpt实际上提示了第21组数据可能有问题，然后发现n可以分解，然后就没有然后。

5.4.22 RSA Backdoor Viability

这题我并没有解出来，只是因为n可以在网站中被分解而拿到了flag，留待日后解决。PADDING

5.4.23 RSA Backdoor Viability

观察可知m^3与n长度差不多，所以是小指数明文爆破，但是还有个padding，也就是实际的$m_0 = m * padding$那么我们求出$padding^3$的模逆元就可以算出$m^3 \equiv mod\ n$，然后爆破即可。

```
from Crypto.Util.number import long_to_bytes
from gmpy2 import invert, iroot
```

```
# n,c,e见output
padding = 16 ** 114
d = invert(padding ** e, n)
m_3 = (c * d) % n
for i in range(1000000):
    if iroot(i * n + m_3, e)[1] == True:
        print(long_to_bytes(iroot(i * n + m_3, e)[0]).decode())
```

SIGNATURES PART 1

5.4.24　Signing Server

将发过来的加密后的信息发回去让它解密就行（doge）。

第6章　后记

历时一月有余终于完成了这篇学习笔记的整理，也算收获颇丰。

由于时间仓促，写作水平有限，本人才疏学线，错漏之处在所难免，欢迎批评指正。我也将在日后不定期修改。

参考文献

[1] Douglas R.Stinson，《Cryptography Theory and Practice Third Edition》

[2] FlappyPig战队著，《CTF特训营》