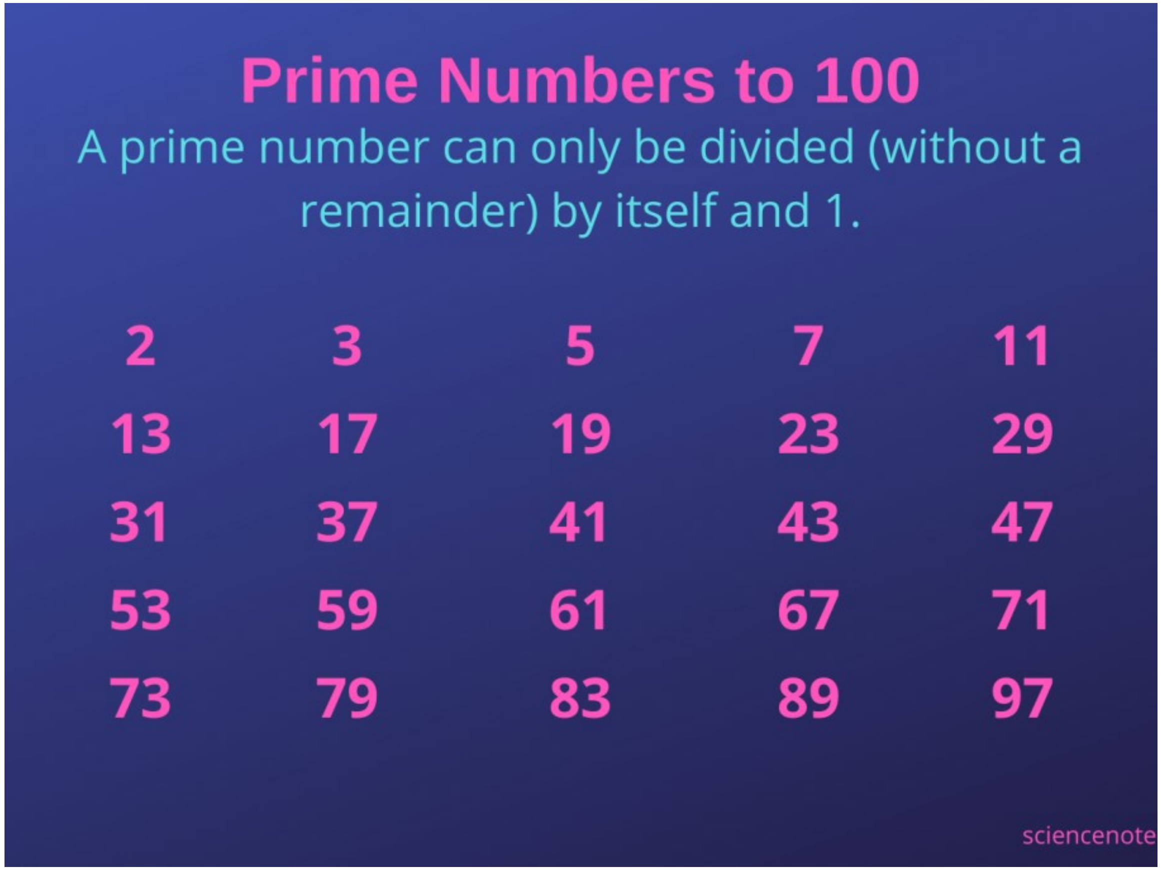
m1z0r3CTF2021 Writeup

作った問題

- 15Prime
  - warmup
- Long Island
  - easy
- 1024Prime
  - medium
- SqUArE
  - hard


Prime Numbers to 100
A prime number can only be divided (without a remainder) by itself and 1.
2 3 5 7 11
13 17 19 23 29
31 37 41 43 47
53 59 61 67 71
73 79 83 89 97
sciencenote

15Prime

15Prime

- Prime numbers are prime in this problem.

- ファイル
  - chal.py
  - output.txt

15Prime

- flagを整数にして、1とANDを取る
  - 最下位ビットが1なら1，0なら0
  - その後1つ右シフトする(=2で割る)

- if文が通ったら
  - 512ビットの素数がoutputに記述される

- 通らなかったら
  - 2つの256ビットの素数の積がoutputに記述される
  - つまり、**合成数**

- つまり、素数かどうか判定できればOK

```python
from secret import FLAG
from Crypto.Util.number import *

flag = bytes_to_long(FLAG)
f = open("output.txt","w")

while 0 < flag:
    if flag&1:
        r = getPrime(512)
    else:
        r = getPrime(256)*getPrime(256)
    f.write(str(r)+"\n")
    flag >>= 1
```

15Prime

- 「python 素数判定 モジュール」 などでググる

- SympyやPyCryptoなどを使う
  - isprime, isPrime関数

- output.txtにはflagの下位ビットから順に処理がされているので
  復元時に注意

m1z0r3{There_are_many_module5_that_1mplement_the
_isprime_funct1on}

Long Island

Long Island

- Have you ever heard of a Japanese comedian named Long Island?

- ファイル
  - chal.py, output.txt

Long Island

- flagのm1z0r3 formatを除いた部分を整数(2進数表現)にする
  - FLAGとする

```python
from secret import flag
from Crypto.Util.number import *
import random

assert flag[:7] == "m1z0r3{"
assert flag[-1] == "}"

FLAG = bin(bytes_to_long(flag[7:-1].encode()))[2:]

key = bin(random.getrandbits(31))[2:]
assert len(key) == 31

ciphertext = []

for i in range(len(FLAG)):
    ciphertext.append(ord(FLAG[i])^ord(key[i%len(key)]))

print(ciphertext)
```

- 31ビットのkeyを生成

- FLAGのi文字目とkeyのi mod 31文字目をXORしたのが暗号文

Long Island

- 31ビットをbrute forceするのは不可能
  - brute forceできるのは大体100万程度まで
  - $2^{20} = 100$万
  - 2^{20}回の処理に10秒かかったとすると、
    $2^{31} = 2^{11} \times 2^{20} = 2048 \times 10[sec] \cong 6[h]$
    - できないわけじゃないが…

- FLAGの内容が完全に分からないとしても、部分的に分かることはないか…？

Long Island

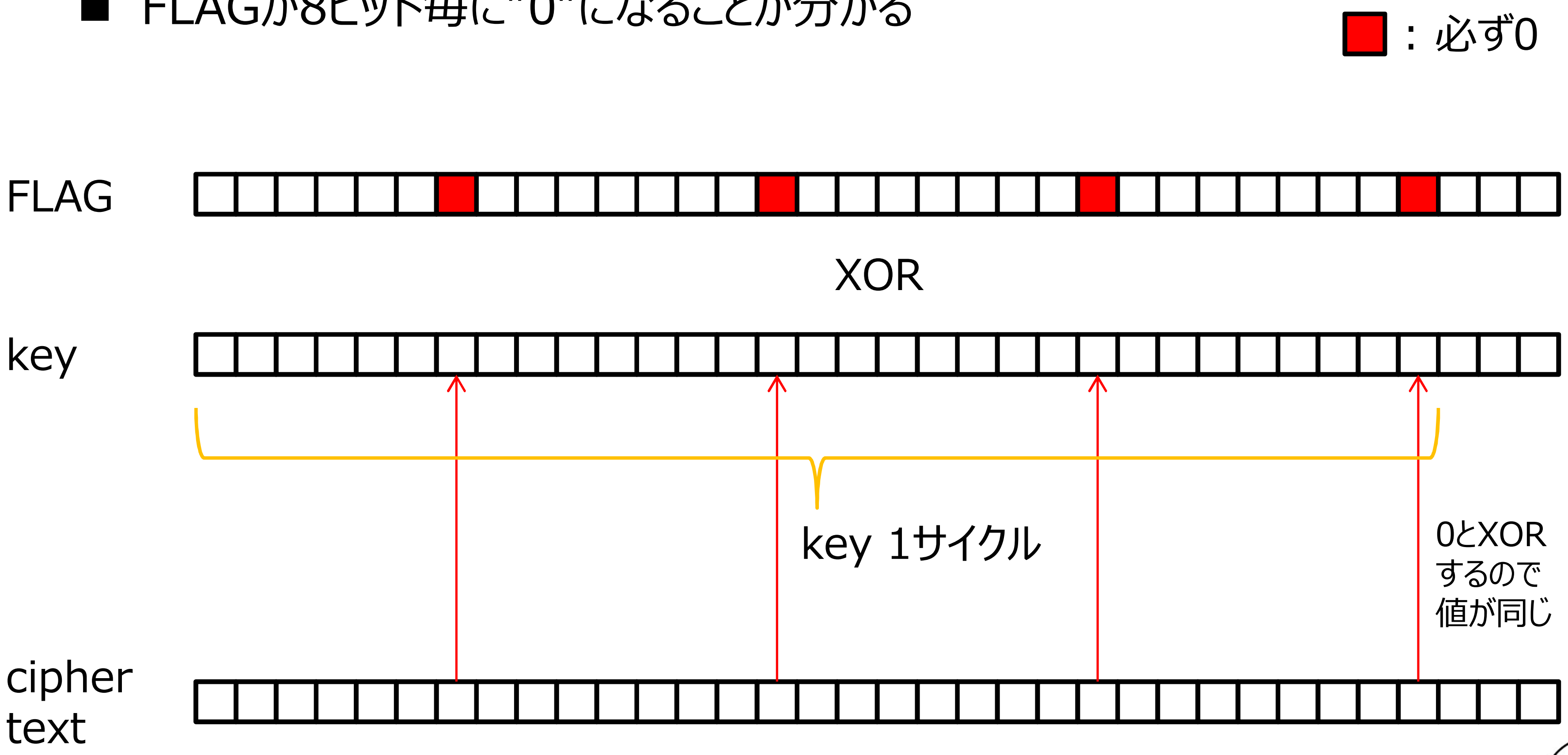
- ASCIIの可視文字列は7bitで表現できるが、
  8bit = 1byte分の値を確保している
  - bin(bytes_to_long(b”rs”)) = “0b111001001110011”
  - 下位8kビット目は必ず0になる

- この性質を使ってkeyを復元する

- FLAGが必ず”0”になるのは何ビット目か？
  - ciphertextの長さは470 = 6 mod 8
  - 最初の文字の上位2bitが切られている
  - 8k+7ビット目が必ず”0”になる

Long Island

- FLAGが8ビット毎に”0”になることが分かる



Long Island

- keyの長さ**31**と必ず0になる**8**ビット毎
  - 互いに素

- keyは繰り返し使われる
  - FLAGの0文字目と31文字目はkeyの0文字でXORする
  - どのkeyのi文字目も必ず0になるFLAGの8k+7文字目とXORする

- 以上を繰り返してkeyを復元する

**m1z0r3{009->59->31_
This_is_Seiya_Matsubara's_backnumber_transition}**

余談

- keyの長さ：31
- 31：巨人、松原聖弥選手の背番号
- 松原聖弥選手の兄：ロングアイランドの松原ゆい
- 59：松原聖弥選手のひとつ前の背番号＆flagの長さ(format除いたもの)

1024Prime

1024Prime

- nc
- ファイル：server.py

1024Prime

- 公開鍵(e, n)と、暗号文cが渡される

- 任意の暗号文を復号してくれる
  - 2048回まで

- 復号された値に対して、下位 i bit目(i:0~1023) が立っている場合、i+1番目に小さい素数のすべての積を1024で割った余りの値が送られる

```python
while cnt < 2048:
    cnt += 1
    send_msg(s,"Input ciphertext > ")
    try:
        ct = int(s.recv(4096).decode().strip())
        m = pow(ct,d,n)
        binm = bin(m)[2:]
        binm = "0"*(1024-len(binm))+ binm
        ret = 1
        for i in range(1024):
            if binm[i] == "1": ret *= Primes[i]
            ret %= 1024
        send_msg(s,"Here you are : "+str(ret)+"\n\n")
```

1024Prime

- 返り値から復号文を再現するのは不可能
  - 返り値は1024通り
  - 復号文は2^{1024}通り
  - 2^{1024}をすべて試す時間はないし、一つの返り値に対して何通りもの復号文が考えられる

- しかし、最下位bitが立っているかは一目瞭然である
  - 最下位bitが立っている場合、偶数を1024で割った余りが返り値なので、**偶数**
  - 最下位bitが立っていない場合、奇数を1024で割った余りが返り値なので、**奇数**

- この情報だけで平文が分かる手法は…

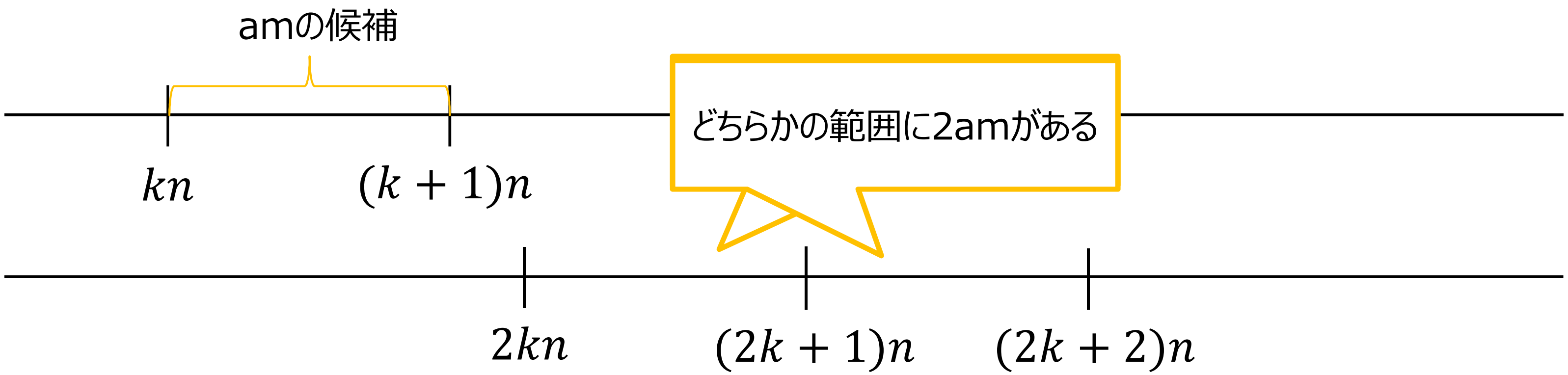
1024Prime

- これだ

LSB Decryption Oracle Attack

- 以下のことが分かっている
  - $kn < am < (k+1)n$

- 以下のことがLSBから分かるようになる
  - $2kn < 2am < (2k+1)n \;\; or \;\; (2k+1)n < 2am < 2(k+1)n$
  - $kn < m < \frac{(k+1)n}{2} \quad or \quad \frac{(k+1)n}{2} < m < (k+1)n$
  - **候補が半分**になる



LSB Decryption Oracle Attack

- 第一ステップ

- $0n < m < 1n$ というのは明らか
  - $0n < 2m < 2n$ となる

- $0n < 2m < 1n$ もしくは $n < 2m < 2n$ かを求める

- $c' = 2^e c \ mod \ n$ を復号してもらう
  - $c = m^e mod \ n$
  - $c' = 2^e c = 2^e m^e = (2m)^e \ mod \ n$
  - c' を復号してもらうと $2m \ mod \ n$ になる
  - つまり返り値は $2m \ mod \ n$

LSB Decryption Oracle Attack

- (返り値) $= 2m - kn$ ただし、$k = 0\ or\ 1$
  - $2m < 2n$ より

- 返り値が偶数の場合、$k = 0$となる
  - $2m\ mod\ n = 2m$ となり、$0 < 2m < n$
  - $0 < m < \frac{1}{2}n$

つまりは、$2m$がnを
超えるかどうか

- 返り値が奇数の場合、$k = 1$となる
  - $2m\ mod\ n = 2m - n$ となり、$n < 2m < 2n$
  - $\frac{1}{2}n < m < n$

- 候補が半分となった。

LSB Decryption Oracle Attack

- 次は$4m$がどの範囲になるか

- $m' = 2m \bmod n$とした時に、$2m' \bmod n$のLSBを調べる
  - $kn < 2m < (k+1)n$ということが分かっている
  - $2m = kn + m'$と言える

- $2m'$がnを超えてた時
  - $4m = 2(2m) = 2(kn + m') = 2kn + 2m'$
  - $n < 2m' < 2n$であることが分かったので、この式と上の式を合わせる
    - $n < 2m' < 2n$
    - $2kn + n < 2kn + 2m' < 2kn + 2n$
    - $(2k+1)n < 4m < (2k+2)n$

範囲が半分になった

LSB Decryption Oracle Attack

- $2m'$がnを超えていない時
  - $4m = 2(2m) = 2(kn + m') = 2kn + 2m'$
  - $n < 2m' < 2n$ であることが分かったので、この式と上の式を合わせる
    - $0 < 2m' < n$
    - $2kn < 2kn + 2m' < 2kn + n$
    - $2kn < 4m < (2k + 1)n$

範囲が半分になった

- $4m$の範囲が分かったので、$m' = 4m \ mod \ n$ とした$2m' mod \ n$ のLSBを利用して$8m$の範囲を求める

LSB Decryption Oracle Attack

- 一般化すると
  - $kn < 2^i m < (k+1)n$ が分かっている
  - $c' = \left(2^{i+1}\right)^e c \bmod n$ を復号してもらい偶奇を知る
    - $m' = 2^i m \bmod n,\ 2^i m = kn + m'$
  - 偶数なら (LSB = 0)
    - $2m' < n$
    - $2kn < 2kn + 2m' < (2k+1)n$
    - $2kn < 2^{i+1} m < (2k+1)n$

    全ての辺に$2kn$を足す
  - 奇数なら (LSB = 1)
    - $n < 2m' < 2n$
    - $(2k+1)n < 2kn + 2m' < (2k+2)n$
    - $(2k+1)n < 2^{i+1} m < (2k+2)n$

    全ての辺に$2kn$を足す
  - 範囲が半分になった

LSB Decryption Oracle Attack

- 簡単に言うと…

- 復号文が[2のべき乗]$m\ mod\ n$ となるようにオラクルに暗号文を投げる

- 復号文をもらって
  - LSBが0だったら、範囲が半分になるよう上限を小さくする
  - LSBが1だったら、範囲が半分になるよう下限を大きくする

1024Prime

- 1024回繰り返して、平文を得る
  - $kn < 2^{1024}m < (k+1)n$ を満たす整数mの種類が数えられる程度になる
  - flagの最後1バイトは b"}" なので適宜修正する

- Pythonで分数を扱う時の注意
  - Fractionモジュールを使う
  - もしくは配列に分子、分母を整数で格納する
  - 浮動小数点型は誤差が出やすい

m1z0r3{Did_you_know_LSB_Decrytion_Oracle_Attack}

SqUArE

SqUArE

- □■

- ファイル : outputS.txt, chalS.py

Challenge S

- eがいつもの値じゃない

- まずはhintから素因数分解できるか考える

- hintの2乗根は存在した
  - つまり、$(p+q)^2 < n$ なのか？
  - だがそれはあり得ない

```python
#chalS.py

from secretS import p,q,mes
from Crypto.Util.number import *

n = p*q
e = 8750
m = bytes_to_long(mes)
c = pow(m,e,n)
hint = pow((p+q),2,n)

print(f"n = {n}")
print(f"e = {e}")
print(f"c = {c}")
print(f"hint = {hint}")
```

Challenge S

- $(p+q)^2$ を展開する
  - $(p+q)^2 = p^2 + 2pq + q^2 > pq\ (= n)$

- つまり、hintの2乗根は$p+q$ ではない
  - たまたま2乗根が存在していました

- $p^2 \cong q^2 \cong n$ と近似すると $(p+q)^2 = 4n$ となる
  - hint+4n あたりに2乗根が存在するかチェック

- 見つけたら、**解と係数の関係**を使って素因数分解を行う

Challenge S

- eがいつもの値じゃない

- eが(p-1), (q-1)と互いに素でないので
いつもみたいな復号ができない

- そういう時は**カーマイケルの定理**
  - 「e p-1 not coprime」などでググる
  - https://blog.y011d4.com/20201026-not-coprime-e-phi/

```python
#chalS.py

from secretS import p,q,mes
from Crypto.Util.number import *

n = p*q
e = 8750
m = bytes_to_long(mes)
c = pow(m,e,n)
hint = pow((p+q),2,n)

print(f"n = {n}")
print(f"e = {e}")
print(f"c = {c}")
print(f"hint = {hint}")
```

復号方法は
備考まで

Challenge S

- ここに注目

実は次のことが成立しているとき、m の候補を e 個に絞ることができます。

- e は素数
- $p-1 \equiv 0 \mod e$ かつ $p-1 \not\equiv 0 \mod e^2$
- $q-1 \not\equiv 0 \mod e$

- 今回は違う
  - eは素数でないし、p-1 = 0 mod e でもない
  - e = 7であれば、上記の条件が成り立つ

- よって、以下のことを行う
  - $c = c1^{1250}\ mod\ n$ となる$c1$を求める
  - $c1 = m^7\ mod\ n$ となるmを求める

$$c = (m^7)^{1250} = m^{8750}\ mod\ n$$

Challenge S

- $c1$を求める段階でも問題が
  - e′ = 1250は p-1 と q-1 と互いに素ではない
  - ただし、p-1 と q-1どちらも5の倍数ではない

- 以上より、$c1$を求める段階をさらに分解する
  - $c = c2^{625}\ mod\ n$ となる$c2$を求める
  - $c2 = c1^{2}\ mod\ n$ となる$c1$を求める

- $c2$を求める段階は、e=625としたいつもの復号で可能である

Challenge S

- $c = m^2 \ mod \ n$ という形、見覚えないだろうか…

Encryption [edit]

A message M can be encrypted by first converting it to a number $m < n$ using a reversible mapping, then computing $c = m^2 \bmod n$.

Decryption [edit]

The message m can be recovered from the ciphertext c by taking its square root modulo n as follows.

1. Compute the square root of c modulo p and q using these formulas:

$$m_p = c^{\frac{1}{4}(p+1)} \bmod p$$
$$m_q = c^{\frac{1}{4}(q+1)} \bmod q$$

2. Use the extended Euclidean algorithm to find y_p and y_q such that $y_p \cdot p + y_q \cdot q = 1$.
3. Use the Chinese remainder theorem to find the four square roots of c modulo n:

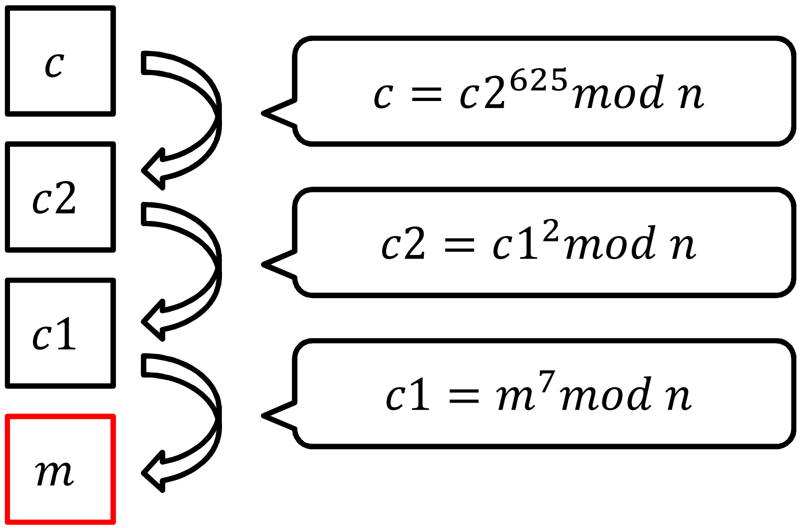
$$r_1 = (y_p \cdot p \cdot m_q + y_q \cdot q \cdot m_p) \bmod n$$
$$r_2 = n - r_1$$
$$r_3 = (y_p \cdot p \cdot m_q - y_q \cdot q \cdot m_p) \bmod n$$
$$r_4 = n - r_3$$

Rabin暗号

Challenge S

- 調べると、p = q = 3 mod 4 なら復号できる
  - 本問題に合致する
  - 復号方法は前頁や備考参照

- この問題は以下のように解く



Challenge S

- mの候補が28通り出てくる
  - $c2$から$c1$を求める段階で、4通りの$c1$が出てくる
  - $c1$からmを求める段階で、一つの$c1$から7通りのmが出てくる

- mの候補を全て出力すると、意味ありげな文字列が…

```
\x80\x12-,5\xf5\x05\xa2\xe4\x02j
b'nc 60.117.58.220 4000'
b'\xda\x03\xf4\xf4,\x14\x9b\xdbi\
```

- 次行ってみよう！！

Challenge U

- 公開鍵(e, n)と秘密鍵dが渡される
- 今よりも小さい秘密鍵をくれといわれる
- 秘密鍵の計算には、nを素因数分解する必要がある

```
I do not like my secret key because it is large.
So, could you make small one instead of me?
e = 65537
d = 34499692108230855438265766533147650098390038014
0942233665
n = 78311385484106593684456343214287113622131751225
9754215751
Give me another small secret key >
```

Challenge U

- $ed \equiv 1 \; mod \; \varphi(n)$ であることを使う
  - $ed - 1 = k\varphi(n)$

- $ed - 1$ を素因数分解して、kと$\varphi(n)$ に分ける
  - 上手くいかないのなら、ncし直す

- $p + q$を求める
  - $n - \varphi(n) = p + q - 1$

- 解と係数の関係より、素因数分解する

Challenge U

- dより小さい秘密鍵の導出方法
  - $ed' \equiv 1 \ mod \ \varphi(n)$ の$\varphi(n)$がより小さいものになれば秘密鍵が小さくなる可能性

- いつぞやの佐古の勉強会資料を遡る
  - $\lambda(n) = lcm(p-1, q-1)$
  - $ed' \equiv 1 \ mod \ \lambda(n)$ でOK

- dが $\lambda(n)$より大きいなら、$d' = d - \lambda(n)$ でもOK
  - $1 = ed = e(d' + \lambda(n)) = ed' \ mod \ \lambda(n)$

- d'をサーバに送ったら、次のnc先が渡される

```
You cleared challenge U! Here is message : nc 60.117.58.220 3000
```

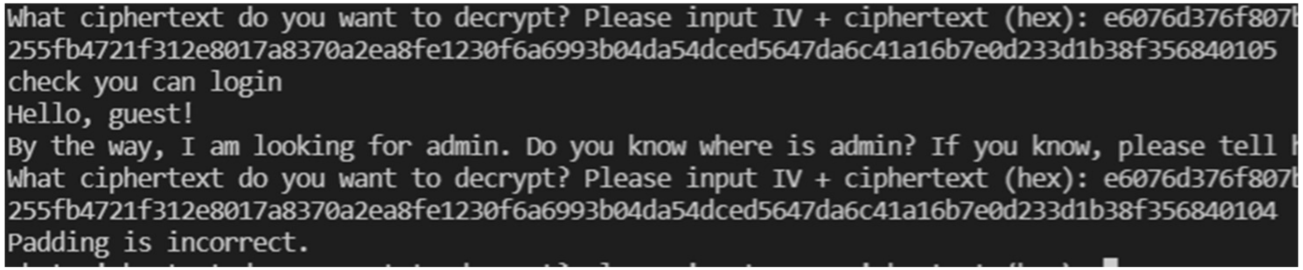
Challenge A

- AES暗号のCBCモードを使っているらしい
- 任意の暗号文を復号してくれるもよう
- 試しにもらった暗号文を復号してもらうと、”guest”でログインできる

```
Welcome to challenge A!
Can you login my system? We use AES-CBC mode.
Here, ciphertext: 2b1aa297f2a13047d953518a01102c6a90d61388255fb4721f312e8017a8370a2ea8fe1230f6a6993b04da5
IV : e6076d376f807b38fbfb430284aaec5a
What ciphertext do you want to decrypt? Please input IV + ciphertext (hex): e6076d376f807b38fbfb430284aae
255fb4721f312e8017a8370a2ea8fe1230f6a6993b04da54dced5647da6c41a16b7e0d233d1b38f356840105
check you can login
Hello, guest!
By the way, I am looking for admin. Do you know where is admin? If you know, please tell him login this
```

Challenge A

- どうやら"admin"にログインしてほしいみたい
- 暗号文を少し変えて復号してもらうと、Paddingが違うと怒られる



- **Padding Oracle Attack**ができると判断

Challenge A

- Decryption Attackを行って、平文取得

```
plaintext m = b'U3FVQXJFIHNlY3JldCBzeXN0ZW06IHVzZXJuYW1lID0gZ3Vlc3Q=\x0c\x0c\x0c\x0c\x0c\x0c\x0c\x0c\x0c\x0c\x0c\x0c'
```

```
>>> base64.b64decode(b'U3FVQXJFIHNlY3JldCBzeXN
b'SqUArE secret system: username = guest'
>>>
```

- その平文を改ざんして最後を"admin"にしたものの暗号文をEncryption Attackで作成する

```
Hello, admin!

Oh, admin! Here is next challenge message!

https://github.com/ksbowler/m1z0r3CTF_2021
```

- Padding Oracle Attackについては参考資料にて

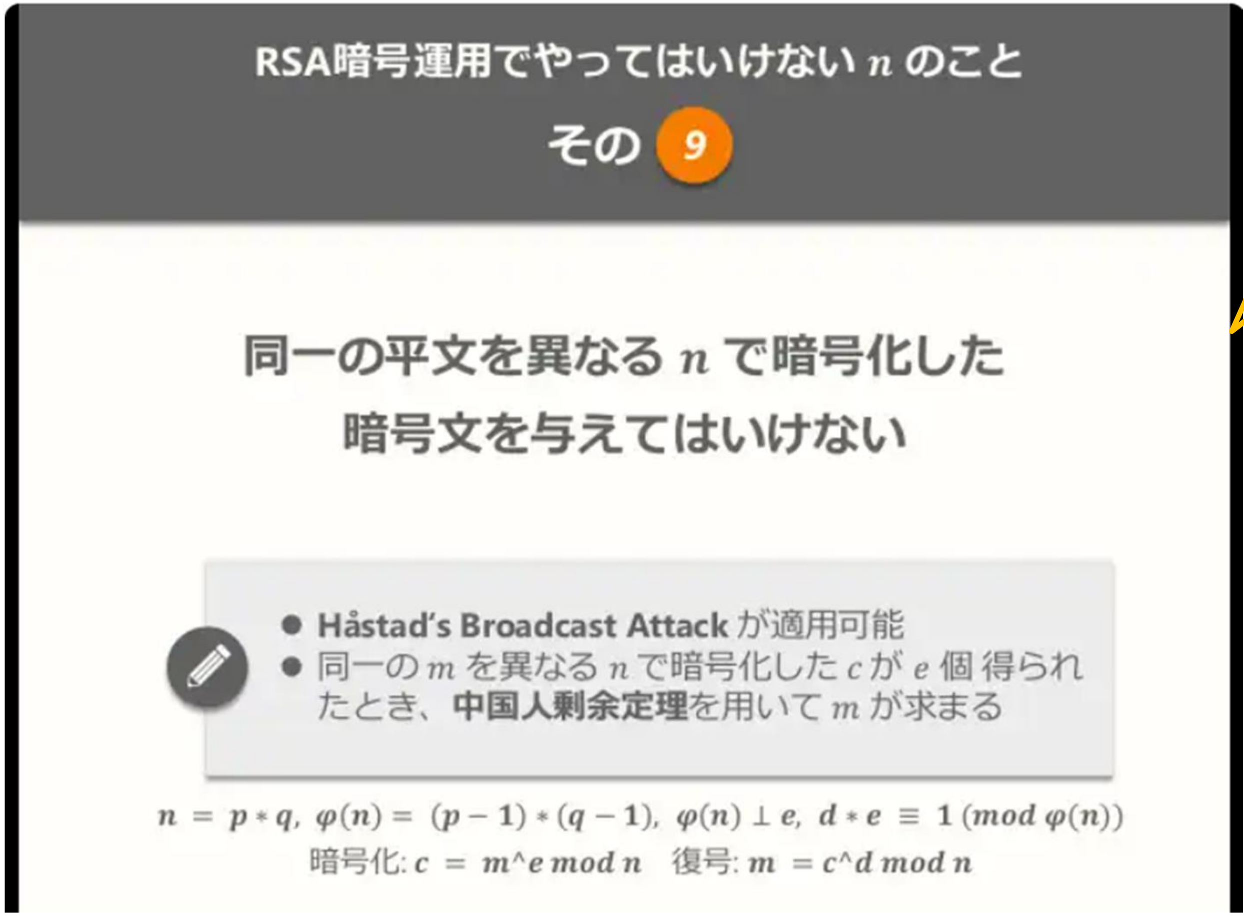
Challenge E

- githubにアクセスすると、2つのファイルが与えられる

- chalE.pyをよく読むと以下のことがわかる
  - n_listにはp, qの積が格納されていて、いくつかある
  - m_listには2進数で値が格納されている
  - m_listの中に含まれている平文の種類数がeとなる
  - m_listの各要素(平文)において、n_listの各要素をnとした暗号文を生成する
    - その値がc_listに格納されている

Challenge E

■ 一つの平文に対して、e個の暗号文が作られているので



復号方法は
備考まで

Challenge E

- 復号してlong_to_bytesしてもわからん
  - 上二つはすべてビットが立っている
- 復号過程でe乗根が存在するか確認するが存在していたので間違ってはいなさそう

```
b'\x1f\xff\xff\xff\xff\xff\xff\xff\xff\xff'
b'\x1f\xff\xff\xff\xff\xff\xff\xff\xff\xff'
b'\x18\x00\x01\x8f\x83\xff>0\x00\x03'
b'\x18\x00\x01\x8f\x83\xff>0\x00\x03'
b'\x18\x00\x01\x8f\x83\xff>0\x00\x03'
b'\x18\xff\xf1\xf3\x83\xe7>1\xff\xe3'
b'\x18\xff\xf1\xf3\x83\xe7>1\xff\xe3'
b'\x18\xc01\xff\xe0\xff\x07\xf1\x80c'
b'\x18\xc01\xff\xe0\xff\x07\xf1\x80c'
b'\x18\xc01\xff\xe0\xff\x07\xf1\x80c'
b'\x18\xc01\xf0\x7f\xf8\xf9\xf1\x80c'
```

Challenge E

- 元は2進数でm_listに格納されていた
  - なにか模様に見える

```
1111111111111111111111111111111111111111111111111111111111111111111111111111111
1111111111111111111111111111111111111111111111111111111111111111111111111111111
1100000000000000000000110001111100000011111111110011111000110000000000000000000011
1100000000000000000000110001111100000011111111110011111000110000000000000000000011
1100000000000000000000110001111100000011111111110011111000110000000000000000000011
1100011111111111111000111110011100000011111001110011111000110001111111111111100011
1100011111111111111000111110011100000011111001110011111000110001111111111111100011
1100011000000000110001111111111111000000111111110000001111111000110000000001100011
1100011000000000110001111111111111000000111111110000001111111000110000000001100011
1100011000000000110001111111111111000000111111110000001111111000110000000001100011
1100011000000000110001111100000011111111111110001111100111110001100000000001100011
1100011000000000110001111100000011111111111110001111100111110001100000000001100011
1100011000000000110001100000111001111100000000001100011000110001100000000001100011
1100011000000000110001100000111001111100000000001100011000110001100000000001100011
1100011000000000110001100000111001111100000000001100011000110001100000000001100011
1100011111111111111000111110011100111001110011100000000111110001111111111111100011
1100011111111111111000111110011100111001110011100000000111110001111111111111100011
1100000000000000000000110001100011000110001100011000110001100000000000000000000011
1100000000000000000000110001100011000110001100011000110001100000000000000000000011
1100000000000000000000110001100011000110001100011000110001100000000000000000000011
1111111111111111111111100011000000111001111111110011100111111111111111111111111111
1111111111111111111111100011000000111001111111110011100111111111111111111111111111
1100011111000000000000000000000000011111000111111100000000000000000110001111111100011
1100011111000000000000000000000000011111000111111100000000000000000110001111111100011
1100011111000000000000000000000000011111000111111100000000000000000110001111111100011
1111111111111111111111000000001111111110000001100000000000000000000000110001111100111111
1111111111111111111111000000001111111110000001100000000000000000000000110001111100111111
1111100111001111100011000110001111111100011000110001111111111100000110000000000011
1111100111001111100011000110001111111100011000110001111111111100000110000000000011
1111100111001111100011000110001111111100011000110001111111111100000110000000000011
1111100111000000000111000000111111000000110000001110000001111100111001110000000000000011
1111100111000000000111000000111111000000110000001110000001111100111001110000000000000011
1100011111111111111000111110011100000000000000111110000000000110001100000011111111111
```

Challenge E

- 問題名はSqUArE
- 全ての平文のbit長が同じ、且つ平文の個数が[平文のbit長]
  - 平文を一行に一つ2進数で出力すると、**正方形**になる
- 問題は、Challenge[S, U, A, E] があった
- なぜqとrが無い…？
- q, r … ?

Challenge E

- **QR code**だ！！！
- PILを使って読み込むのでもいいが面倒くさい
- “1”を”＃＃”に、”0”を”　”(空白2つ)にするなどして見やすくする
- あとはスマホのカメラなどで読み込む

m1z0r3{sQuaRe_ha5_many_mean1ngs_maybe}

参考資料

- Challenge S 類題
  - https://partender810.hatenablog.com/entry/2021/10/30/120348
- Challenge U 参考　2021/10/18
  - https://m1z0r3-cloud.nsl.cs.waseda.ac.jp/index.php/apps/files/?dir=/m1z0r3/Crypro%E5%8B%89%E5%BC%B7%E4%BC%9A/20211018(sako)&fileid=67783
- Challenge A (Padding Oracle Attack)
  - https://partender810.hatenablog.com/entry/2021/06/08/225105

備考

Rabin暗号 復号

- 英語版Wikiの方がわかりやすい
  - https://en.wikipedia.org/wiki/Rabin_cryptosystem

- $c = m^2 \; mod \; n$ を解く
  - nは素因数分解できている前提

- 以下の式よりm_p, m_q を求める
  - $m_p = c^{\frac{p+1}{4}} \; mod \; p \; , \;\; m_q = c^{\frac{q+1}{4}} \; mod \; q$

Rabin暗号 復号

- ユークリッドの互除法より、y_p, y_q を求める
  - $y_p p + y_q q = 1$
  - gmpy2.gcdextが便利

```
g,yp,yq = gmpy2.gcdext(p,q)
```

- 以下のr_iが平文の候補

$$
\begin{aligned}
r_1 &= (y_p \cdot p \cdot m_q + y_q \cdot q \cdot m_p) \bmod n \\
r_2 &= n - r_1 \\
r_3 &= (y_p \cdot p \cdot m_q - y_q \cdot q \cdot m_p) \bmod n \\
r_4 &= n - r_3
\end{aligned}
$$

カーマイケルの定理

- 先ほどのサイト参照

復号方法

- $\lambda = (p-1)(q-1)/\gcd(p-1, q-1) = \mathrm{lcm}(p-1, q-1)$
- $L \equiv k^{\lambda/e} \mod n$ (k は位数が λ となる任意の整数)
- $d \equiv e^{-1} \mod \lambda/e$

としたとき、$c^d L^i \mod n \ (0 \leq i < e)$ が m の候補となります。

- kには、最初2を代入していいのが出なかったら3を試す、という感じ
  - 正直、位数をよく分かっていない

Hastad's Broadcast Attack

- ①
  - $m^3 \equiv c_1 \; mod \; n_1$
  - $m^3 \equiv c_2 \; mod \; n_2$
  - $m^3 \equiv c_3 \; mod \; n_3$

- ②
  - $m^3 > \max(n_1, n_2, n_3)$
  - $m < \min(n_1, n_2, n_3)$ より
  - $m^3 < n_1 n_2 n_3$

← e = 3の場合

本問題の場合、
一つの平文に対して
30個の式ができる

Hastad’s Broadcast Attack

- ①
  - $m^3 \equiv c_1 \, mod \, n_1$
  - $m^3 \equiv c_2 \, mod \, n_2$
  - $m^3 \equiv c_3 \, mod \, n_3$

- n1で割るとc1が余りで、n2で割るとc2が余りで、n3で割るとc3が余りとなる数

Hastad’s Broadcast Attack

- ①
  - $m^3 \equiv c_1 \, mod \, n_1$
  - $m^3 \equiv c_2 \, mod \, n_2$
  - $m^3 \equiv c_3 \, mod \, n_3$

- n1で割るとc1が余りで、n2で割るとc2が余りで、n3で割るとc3が余りとなる数
  - ➢ m^3

- 中国人剰余定理と同じ形

Hastad’s Broadcast Attack

- from sympy.ntheory.modular import crt
  - Message, n = crt(nのlist, cのlist)
  - これで中国人剰余定理が使える

- 最後のgmpy2.irootで3乗根かを確認