腾讯极客挑战赛第一期：解开一道即将尘封十几年的封印 writeup

原创

kang0x0 于 2022-03-25 16:26:27 发布 20 收藏

分类专栏： 腾讯极客技术挑战赛 文章标签： 安全

本文链接：https://blog.csdn.net/kang0x0/article/details/123702937

文章目录

腾讯极客挑战赛第一期：解开一道即将尘封十几年的封印 writeup

简要说明

- 赛题链接：腾讯极客挑战赛第一期：解开一道即将尘封十几年的封印
- 题目代码如下。

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
from Crypto.Cipher import AES
import base64
import time
import gzip
from hashlib import md5
import sys
import io
sys.stdout = io.TextIOWrapper(sys.stdout.detach(), encoding='utf-8', line_buffering=True)
```

```
sys.stdout = io.TextIOWrapper(sys.stdout.detach(), encoding='utf-8', line_buffering=True)

def Decrypt(key:str, text:str) -> str:
  if len(key) < 32: key += ' ' * (32 - len(key))
  elif len(key) > 32: key = key[0:32]
  cipher = AES.new(bytes(key,encoding='utf-8'), AES.MODE_CBC, bytes(AES.block_size))
  return str(gzip.decompress(bytes.strip(cipher.decrypt(base64.b64decode(text)))), encoding='utf-8')


def Pass(id, priv_key):
  prefix = str(id) + str(int(time.time()))
  pub_key = prefix + md5(bytes(prefix + priv_key, 'utf8')).hexdigest()
  print('恭喜通过第%d关,通关公钥:%s' % (id, pub_key))

key=input('1+1=')
exec(Decrypt(key,'JIvH7KUKFAKDu6ZfRjsV9VsCODat2VbDd6S+QAGKEXtGlSxvhUIhqHfXq/1EhGohqhFelniKn3294DpzdccOhP6KcQQPxp
GVgKcQJfezn+4JA4Aq0rvWkVoYew8OkRCt2/7MmgVwLCxlqhIrI5SvibCg2Yg0nBs/qe+7rI2EcC16ncIiBICvQFIvewAsYLcIEHFFdbzkM2nwfj
xFnQ1bqgchYMm0lsKvztSAxxRS6ZFrdZqNb3u8Iyg6DB1vRu2BZFu5ed3E0g926LASeliCxvltvE5EJaJfJtquFAMeJxlcDTEkRdWbdoi5zbB2UK
7ZM+i+STJPK+QKo0MEMAm+pkXmm0ZYttEYXDSqJHoutOVGX73EHnsBtGSYqs20UVHT5AbFXu8adbUtM5eqWJ5NRy8spXVnd/hOZo/qoS/Yp6LAKw
WccC/J1As//SDpm+gsYENoKVgGoqJFStWccrqk6pWGIwEwimUq2tXaTsfCbHYCNT+AOrWYD0w6c3LJdFj38PrZSYjEceJHFeP7bdX2u5JmXlXKrZ
gpDNVP/RnQS1Zhw76ZTid31IPprHVHD1indT21WapbtdVuhDijAYpAFvzVmjeFPXjaUuAZwJw9voW/jg9Ucfe0OScMs82xVTW0EfBqPpM2WH+OXj
C+xZUrrlqkuqG67qaf66Lhl+uSuuGinTIbzaMnlY8CyNpRBbJyHpu4/keDWZC2n0C5DCdvmWIQHtM0UJs0v4MICgu74Rrf11tmuUvKb4htLMTGT3
BDjELZQvejWqMNjKods8W+B62hKYqLJDyJEsxjGe1uZWdmyZnm4oPLwzpJLlOZqIUL+uJkm7/nCkqadPdRQT/80xXz+K4btjaNkiKmTPSBtnCs3c
lWH1ZDHehMTZXu6Md2Y9TUjVXoEB7f96ZmWmuttFuLBnLpT9FsOxxHL1XBXSusgltORLgJx7t2zrcFJr+z8Uw3fyiN6XiR/YdbMhhUucgroPLhJB
0Z6g0h5pdKjmyHsXzQ9k9PA8hdXHzME4MG7rdi7IsHPMC56PPoxenrkNLnFrcwxJ4vmVPhXHqljKo0PrtGsfFHw3Yy5/MqOmz5ZSN9F92gZQiHZw
hKLXW/HNGnOexEONDCSccDch7Nt7ztqlcA3fygD6Kx8/N+YNTtiudlw6ZG3FzCaZusn9JQsswrhYMN2lWCSSB+JB2Ol1yOHwIGRKCJ+cj6XShojG
/KHbfDahNt4GPZi7fK+8kIUir+9KQ8PqEFi1K9N868oqlY1JN85LhA55WPdvVlTAe8o7XQCVYM31ce9iM/ZCRLC6uAu/EVK1aju4zgMumxQumfSD
n4J3m80R4WANDvyPSmqqhB950TqarXHc9ni9g91wp6OqmZcs43Mtwyj5DLpITc1AZTGagiLDC8ChDZJQ7v2o5Hegf4iPdTSB4j8bMkRYDOAjLutS
ix4tqA5uDt7z069UPIhNUSFWOhGkN2jzUqoITNbOx1Icxbj4YPsiZ3bT3DUXoEzAtjf6JW8N9X3iItG9kz8LqdnkpUmOtaMlDwTXnbQC1/gkFZKu
CPK0Nf4PXiEmWLUcaajM1mCuKDrTRqaevcqsOXIVw2dODsQQTLysnQaAXlWJv9jYYCpcenvQ9dVGc5XJz7NNzBcy1XmNBrctQuiUvc1v2IkQfKVl
mlEo4OaN0ZkxjQZZUkg3ghyr7dA3qve3VRn6i9ObPC1MmATr5NjXsBoyhDO9nidqZYfRhJamhL5AuCR4Y91PI2h9qapdGbRYJs1WX3d5qZ/wVTt6
dHFAZPwxL7wEHmevLCoGw6Fp8YnxVZGynwsonR37WfQt6BcNYUZMPr4Is9rO79tRmbsOe932VOCi1dZ2eEvEMM5hah6/1fc266Ssu6HHsmkkrwe8
C74QTwduP0vpxD1kX5GSu9jq2Y4Keg5nCRtBlMg2xdIeyyg4CIDX7BYDkmP4Yn/3xczpbB7+PfB80x0qi70u4mfEikdwuasaxkChIEXBBaMAdjUj
7rVfJvasy/hUNZ6tp2AJwwBfLKSLxsKIb7p0E+a/Vz0lJ88u3HHjqiL/UjN6qTV5oWFJcU303Bpbh8wlTRoFU89Jq31GfkPbuifwGEmTgjyzQpg6
AJP0K9wJX3f7C8W2TbEeUA3noWkNtl814jvbovSIB/inK1DWuChLsn9eInyLJ7d7u/OFL/UFPA/C5fvAsS/l+Kwf68ghZRB8ftr/x8b835k2woU2
LWgbi70R3iNVBQ/q04lxYJYImYaHWGRyQCjv4n6WF1c53fN7l9ATuNOwR57Ap7XpEwHSSAeP/kt7pkhM4wp6o17XRYiHjzZI/hv+9LieLPB+uLpt
h1PoL2Lo0w5930Dj/g1gLtJAdowfjyvSjcIUUHwVZOkjmgm/vvEH0pFohWTZr7ZSPkGvXwEEdjocWA/4qNCHSbXXceqDqEaW7w/599WkEKbA5zTw
04c0AsXSrCjPGgm99ZGvIn0/8I7XUdR7uPbw36ybgwjBYCq37jqCDf5wxNp7UhXLLHehn4TtGGlX6v6iwDVU2tWBS3U8BfWRIqTTUtrr+b3U1J2b
Hi2cDmvLS4ym5eci0Kv7XHD9cj2aBj6cPOkXt0kgBNiylVwFJg0bcuNWYOXeN36kj3PIVrSJ7mDqCYT1wupgQT/PlYZpq6uy1YuBS8loSfi0TP3u
Xr5gz4ZKCd5UhA5Dj4qeSYJs2tOkpSOhMQMguZYNHeZrPnHJMRq7I3LqZOAnQ299Y9JEN5YNT2s5PrgqkzzQka4IV9bE3JgxykW66ZJxapHG820a
H9s5RvOMcdJJms/FA/kX0oOiLNrYW450Ec70MPi4ZGzom4tqavSyPj/iYZlVHAt2WIB3zoToIgf4rcjkgshN81tGg33zpIV59j3sWJ7paqEoE7Bs
zOz0193AUML7NC7dJJpJStH+pkGncL91at4eeMplBXUBIuKknrrEti/X4eFvBY8ns0hHH+pI5uv3tyGxdI3GkHpwLRxGlyLR4Wril9VcIqiTMhdc
ag/JS5AByd68RkHkKJScwX7Qb9t1uWsplbQ0SlSvqZgQqNO5Rw126B/ywXPHOLgpUfrgp3EnhJ/3mxdxDF8Lj6GP+nEChzVa4eZ0lZBLsyDJeGI2
rmKKDQLMGZMs+xtLB9kfrIvlvLyTTuSXzlX/EDJ+BEmVlURyELCEDezhWT60Lt2kGJwCp2hl+pzbQh7wc0bbBgWRJwzdD74rZgWlHG8D8wOYlf+o
btM2tjY5DCsxZtiEVatcdnhPqSZI3eIHnLHpfDZu69VMm01FlQwWirtK6cHIJAjXYnQEnj6H90Rp2LczNhzJkzS1vo/sV1N5iHP0Y+NE5Q1kypPH
wTkOc0XdSlh3WIYwiYFtXu5PsLvYqbCcbjaBP6MbbOjTiwE73uMzp3T3hG3VzoqGWCYQFsDYtuz8/3uhHFEMFKjd0dhvV8q7bdCMgfJ8gm9CaEvn
TH4h6Ta/fnermWvkBGveV7hE5lCDknDoKJzNU2giiHZHv77HvQuqnHG2UxLwFWrWNsYtqA8GTUYyxxr7sKxikCKdl079qVDUp99Xb/0CpNx8f1aj
Vg3VWGPHwY7v0BTITax+z/JG8EolLRua9oyb2uCx827/9F6A+D5bmZaKbImeOzejSslLx7lZkA/8cs1JzbdpgBcXP2cHvXmrWutxiLJkDiKgXOEE
/trdSwzYXn5TwWSRCtRx65D3RGKnjA7mPpSpHWmOJz7NpIxgi3CJSGmZAkPp6NjskpIhqPMAD1MjyY6BmlqSXvgNVArNEHegOoZWCwHVgO/0hxM2
hUcSq1f1SPoq1N61qXQvw66DjgCYOLLb47lW3Y9OWWFCtDxnbR9w52xv8XyohW+26c/QGx07Z4Tt4k2Em7gslWSQiqvclL+P0cjVy75uwG0a0ARb
BBADit9QFVFnsZyLQ3qCyTLi73LGRVzD11PsL6se7pRvRWMNmvmiQKw/4SfTaYF1srWpaDxgVwHoF2l2bufgatZufXyGOqMQW1b4Oim943Fobf81
+jhPipKeonMspKrx1S/8iifz7UVXAVh2MebJo8YEQszRg38DzMcK2AxpXFANWA8i2tdVtU++njQXzM655+wblloZYa2s/x8iOO/YMHw4Q4iH5YfI
p602tbOTUdYbTw3avhIC0vBsAzwi1kPOvfZeWXSPfqMChAvBboPPsEmu5ST/RFbWF3Wph/MPjKr548wudh29MRdKDqvTvK8ZCA9ymEIs6/nXyXVr
Pg3WMlVCwuiST+zsd4Aph3G2S051ndEiOqgirG6CVejwGg40YKG4f7jUWxL+Kps69ialit/Fz2+gG5jeZG+PmagxjnYHZtCzrWu4uYV+IQuJXcql
NIFznSTsEsvU2lbQgCbkSp9/CFtZqE4bXz8Oe02/j/rjnSGylT8VlrRa25O64byQYljv6Gvr6kgxcp8FygFcAjMzBaamYZydH5ZnSNBBrzrWeuWP
2NfamUM0eGccSbhf3mWeJjm7O1ybYxAJdLqOTTh3AYE+nzhl9nOoF7QSC4eIIDGO0+PFMCr9IltBaNwx7AmhrIvaAOwyct+tJuDT0EKxPhuNfIJW
NJ6ub3UT7iGB4xPVzIERA1Mue7UuvLdardWhMqAqFhBEDzFwNwM7b/lJsoRPFoc+WJr8isCLLfiGjzZhpuHmzVfMXwCOUvZnzYBUqHsxx4SAJPwk
0PW6qUWkUG3vYCrRb6I/qge9QuYHPTQ5OE9WzQef9HIm7tp6bqywArRM+b7Mm0ldUz/ugebDo9cKGQqm4I3rBZ0FXh/VMdxbH6e/+0snAWdmL36V
uLgXAVHko1hPsHe3PO/DVQhUXQQITMMJ2yUajWCmGHqFIyS9gqVqG9E9WdTSkmxs+2h4g+sk5OuPKdczvzm9Yf5oA49lksQuJcWD3M0MaXnvH07x
wEsQuJiRWdo0JzPXA0OuMcQ1GPUV5E/rMiNn4yjRPP/HAFP7LlfKmkguFfcOsYyXhkNQ2zow9Q4+F12qXiHJGT5ShL4dZWiSU6PCgAmh/cLqFSD6
```

```
+ILK4wOBRz9gqlck1pocJJazkP8FaXadW6+pfIWSeVSKQcsZDIXySu453ZsNxAtHOp1/TgtQZFpuarIVSGbUIpwqUacoL3NcuxuBhznHVLUp6WVv
xNks4Z5O4wWH4c3tnE7qrx8r0qcVeuFrTRw96ICkDHqWNEr+gZrIlKAed9KIqGqMzjBZK+QtXDMECCXaS0nIab+ZlRNKFpWiqObLKPkSpLKZ5owc
uO7EOudaeI6xc50wa7z6FBNMd2oCS9JWt14bbtMLnPXvZ+iMXMgEP929qnFtKZzeRcvkkvnMbaGrqsb/yiQVX5wan6rUzunAWPdTVgcqJT1Pi54G
/OQxiVlcyvg4/PRAfV+8RLW0qeHhJExUVPIS8mz5fE3MIvLNgBHCqsQe/GnLMBV2aUqH1l5o1WsvVTWYJYWZHKZbxpSixxkx1qLeHO+W2NHGJHL6
rWOJctmVuW9IDusIjeGC/L4t1ZygZlkKgpq848PIhMetJxD9j8Aq6GK3gxlXax7dpQ2y/J53kgHbDEvslD5x6MlswhgWcwC9hDcb/gYYTr8BmrZd
0LtvCzrOJAYsCPObZbZPqOO37gbykhRhJ2FQv0+Lvp+lj/M5OoRmHtrTPjqNaDVmDncSPTIajXjAItkRxJLJboacSeEsGsJvSD0H0xgUhzhOfK0Q
epXXLfzG4aX/ow7we9pOXw3G7ydfdd9iB1yCiIICaW3SAavL2zy/dHMb5/0a0WxMza89pRW8KMZ/GQSxZOS2Ek8fJ954mEbJv8c5ZrzKyC9fbO89
FsZmHimnBNZBlGyNrKckhBywYcHI/k4ytgkWMpFmYiNxV8j0WVmw1NDXuF/FCnRHHnexgRiVoZU8SWtnBWAqz4gZt3Z9ehoGXYKWXjS8eG0bWX6u
eeNYrNKND5b1zXEd3SlN1UTqrtiqa2NKFAht0DlsMxYqweGTBMk4h06w=='))
```

第一题 1+1=?

- 题目提示key=input('1+1=')
- 很自然地去尝试 key='2'，得到答案以及下一关题目。
- 通过Pass函数获取需要提交的公钥即可。

```
# 第一题
key="2"
print(Decrypt(key,'JIvH7KUKFAKDu6ZfRjsV9VsCODat2VbDd6S+QAGKEXtGlSxvhUIhqHfXq/1EhGohqhFelniKn3294DpzdccOhP6KcQQPx
pGVgKcQJfezn+4JA4Aq0rvWkVoYew8OkRCt2/7MmgVwLCxlqhIrI5SvibCg2Yg0nBs/qe+7rI2EcC16ncIiBICvQFIvewAsYLcIEHFFdbzkM2nwf
jxFnQ1bqgchYMm0lsKvztSAxxRS6ZFrdZqNb3u8Iyg6DB1vRu2BZFu5ed3E0g926LASeliCxvltvE5EJaJfJtquFAMeJxlcDTEkRdWbdoi5zbB2U
K7ZM+i+STJPK+QKo0MEMAm+pkXmm0ZYttEYXDSqJHoutOVGX73EHnsBtGSYqs20UVHT5AbFXu8adbUtM5eqWJ5NRy8spXVnd/hOZo/qoS/Yp6LAK
wWccC/J1As//SDpm+gsYENoKVgGoqJFStWccrqk6pWGIwEwimUq2tXaTsfCbHYCNT+AOrWYD0w6c3LJdFj38PrZSYjEceJHFeP7bdX2u5JmXlXKr
ZgpDNVP/RnQS1Zhw76ZTid31IPprHVHD1indT21WapbtdVuhDijAYpAFvzVmjeFPXjaUuAZwJw9voW/jg9Ucfe0OScMs82xVTW0EfBqPpM2WH+OX
jC+xZUrrlqkuqG67qaf66Lhl+uSuuGinTIbzaMnlY8CyNpRBbJyHpu4/keDWZC2n0C5DCdvmWIQHtM0UJs0v4MICgu74Rrf11tmuUvKb4htLMTGT
3BDjELZQvejWqMNjKods8W+B62hKYqLJDyJEsxjGe1uZWdmyZnm4oPLwzpJLlOZqIUL+uJkm7/nCkqadPdRQT/80xXz+K4btjaNkiKmTPSBtnCs3
clWH1ZDHehMTZXu6Md2Y9TUjVXoEB7f96ZmWmuttFuLBnLpT9FsOxxHL1XBXSusgltORLgJx7t2zrcFJr+z8Uw3fyiN6XiR/YdbMhhUucgroPLhJ
B0Z6g0h5pdKjmyHsXzQ9k9PA8hdXHzME4MG7rdi7IsHPMC56PPoxenrkNLnFrcwxJ4vmVPhXHqljKo0PrtGsfFHw3Yy5/MqOmz5ZSN9F92gZQiHZ
whKLXW/HNGnOexEONDCSccDch7Nt7ztqlcA3fygD6Kx8/N+YNTtiudlw6ZG3FzCaZusn9JQsswrhYMN2lWCSSB+JB2Ol1yOHwIGRKCJ+cj6XShoj
G/KHbfDahNt4GPZi7fK+8kIUir+9KQ8PqEFi1K9N868oqlY1JN85LhA55WPdvVlTAe8o7XQCVYM31ce9iM/ZCRLC6uAu/EVK1aju4zgMumxQumfS
Dn4J3m80R4WANDvyPSmqqhB950TqarXHc9ni9g91wp6OqmZcs43Mtwyj5DLpITc1AZTGagiLDC8ChDZJQ7v2o5Hegf4iPdTSB4j8bMkRYDOAjLut
Six4tqA5uDt7z069UPIhNUSFWOhGkN2jzUqoITNbOx1Icxbj4YPsiZ3bT3DUXoEzAtjf6JW8N9X3iItG9kz8LqdnkpUmOtaMlDwTXnbQC1/gkFZK
uCPK0Nf4PXiEmWLUcaajM1mCuKDrTRqaevcqsOXIVw2dODsQQTLysnQaAXlWJv9jYYCpcenvQ9dVGc5XJz7NNzBcy1XmNBrctQuiUvc1v2IkQfKV
1mlEo4OaN0ZkxjQZZUkg3ghyr7dA3qve3VRn6i9ObPC1MmATr5NjXsBoyhDO9nidqZYfRhJamhL5AuCR4Y91PI2h9qapdGbRYJs1WX3d5qZ/wVTt
6dHFAZPwxL7wEHmevLCoGw6Fp8YnxVZGynwsonR37WfQt6BcNYUZMPr4Is9rO79tRmbsOe932VOCi1dZ2eEvEMM5hah6/1fc266Ssu6HHsmkkrwe
8C74QTwduP0vpxD1kX5GSu9jq2Y4Keg5nCRtBlMg2xdIeyyg4CIDX7BYDkmP4Yn/3xczpbB7+PfB80x0qi70u4mfEikdwuasaxkChIEXBBaMAdjU
j7rVfJvasy/hUNZ6tp2AJwwBfLKSLxsKIb7p0E+a/Vz0lJ88u3HHjqiL/UjN6qTV5oWFJcU303Bpbh8wlTRoFU89Jq31GfkPbuifwGEmTgjyzQpg
6AJP0K9wJX3f7C8W2TbEeUA3noWkNtl814jvbovSIB/inK1DWuChLsn9eInyLJ7d7u/OFL/UFPA/C5fvAsS/l+Kwf68ghZRB8ftr/x8b835k2woU
2LWgbi70R3iNVBQ/q04lxYJYImYaHWGRyQCjv4n6WF1c53fN7l9ATuNOwR57Ap7XpEwHSSAeP/kt7pkhM4wp6o17XRYiHjzZI/hv+9LieLPB+uLp
th1PoL2Lo0w5930Dj/g1gLtJAdowfjyvSjcIUUHwVZOkjmgm/vvEH0pFohWTZr7ZSPkGvXwEEdjocWA/4qNCHSbXXceqDqEaW7w/599WkEKbA5zT
w04c0AsXSrCjPGgm99ZGvIn0/8I7XUdR7uPbw36ybgwjBYCq37jqCDf5wxNp7UhXLLHehn4TtGGlX6v6iwDVU2tWBS3U8BfWRIqTTUtrr+b3U1J2
bHi2cDmvLS4ym5eci0Kv7XHD9cj2aBj6cPOkXt0kgBNiylVwFJg0bcuNWYOXeN36kj3PIVrSJ7mDqCYT1wupgQT/PlYZpq6uy1YuBS8loSfi0TP3
uXr5gz4ZKCd5UhA5Dj4qeSYJs2tOkpSOhMQMguZYNHeZrPnHJMRq7I3LqZOAnQ299Y9JEN5YNT2s5PrgqkzzQka4IV9bE3JgxykW66ZJxapHG820
aH9s5RvOMcdJJms/FA/kX0oOiLNrYW450Ec70MPi4ZGzom4tqavSyPj/iYZlVHAt2WIB3zoToIgf4rcjkgshN81tGg33zpIV59j3sWJ7paqEoE7B
szOz0193AUML7NC7dJJpJStH+pkGncL91at4eeMplBXUBIuKknrrEti/X4eFvBY8ns0hHH+pI5uv3tyGxdI3GkHpwLRxGlyLR4Wril9VcIqiTMhd
cag/JS5AByd68RkHkKJScwX7Qb9t1uWsplbQ0SlSvqZgQqNO5Rw126B/ywXPHOLgpUfrgp3EnhJ/3mxdxDF8Lj6GP+nEChzVa4eZ0lZBLsyDJeGI
2rmKKDQLMGZMs+xtLB9kfrIvlvLyTTuSXzlX/EDJ+BEmVlURyELCEDezhWT60Lt2kGJwCp2hl+pzbQh7wc0bbBgWRJwzdD74rZgWlHG8D8wOYlf+
obtM2tjY5DCsxZtiEVatcdnhPqSZI3eIHnLHpfDZu69VMm01FlQwWirtK6cHIJAjXYnQEnj6H90Rp2LczNhzJkzS1vo/sV1N5iHP0Y+NE5Q1kypP
HwTkOc0XdSlh3WIYwiYFtXu5PsLvYqbCcbjaBP6MbbOjTiwE73uMzp3T3hG3VzoqGWCYQFsDYtuz8/3uhHFEMFKjd0dhvV8q7bdCMgfJ8gm9CaEv
nTH4h6Ta/fnermWvkBGveV7hE5lCDknDoKJzNU2giiHZHv77HvQuqnHG2UxLwFWrWNsYtqA8GTUYyxxr7sKxikCKdl079qVDUp99Xb/0CpNx8f1a
jVg3VWGPHwY7v0BTITax+z/JG8EolLRua9oyb2uCx827/9F6A+D5bmZaKbImeOzejSslLx7lZkA/8cs1JzbdpgBcXP2cHvXmrWutxiLJkDiKgXOE
E/trdSwzYXn5TwWSRCtRx65D3RGKnjA7mPpSpHWmOJz7NpIxgi3CJSGmZAkPp6NjskpIhqPMAD1MjyY6BmlqSXvgNVArNEHegOoZWCwHVgO/0hxM
2hUcSq1f1SPoq1N61qXQvw66DjgCYOLLb47lW3Y9OWWFCtDxnbR9w52xv8XyohW+26c/QGx07Z4Tt4k2Em7gslWSQiqvclL+P0cjVy75uwG0a0AR
bBBADit9QFVFnsZyLQ3qCyTLi73LGRVzD11PsL6se7pRvRWMNmvmiQKw/4SfTaYF1srWpaDxgVwHoF2l2bufgatZufXyGOqMQW1b4Oim943Fobf8
1+jhPipKeonMspKrx1S/8iifz7UVXAVh2MebJo8YEQszRg38DzMcK2AxpXFANWA8i2tdVtU++njqXzM655+wblloZYa2s/x8iOO/YMHw4Q4iH5Yf
Ip602tbOTUdYbTw3avhIC0vBsAzwi1kPOvfZeWXSPfqMChAvBboPPsEmu5ST/RFbWF3Wph/MPjKr548wudh29MRdKDqvTvK8ZCA9ymEIs6/nXyXV
rPg3WMlVCwuiST+zsd4Aph3G2S051ndEiOqgirG6CVejwGg40YKG4f7jUWxL+Kps69ialit/Fz2+gG5jeZG+PmagxjnYHZtCzrWu4uYV+IQuJXcq
```

```
P2NfamUM0eGccSbhf3mWeJjm7O1ybYxAJdLqOTTh3AYE+nzhl9nOoF7QSC4eIIDGO0+PFMCr9IltBaNwx7AmhrIvaAOwyct+tJuDT0EKxPhuNfIJ
WNJ6ub3UT7iGB4xPVzIERA1Mue7UuvLdardWhMqAqFhBEDzFwNwM7b/lJsoRPFoc+WJr8isCLLfiGjzZhpuHmzVfMXwCOUvZnzYBUqHsxx4SAJPw
k0PW6qUWkUG3vYCrRb6I/qge9QuYHPTQ5OE9WzQef9HIm7tp6bqywArRM+b7Mm0ldUz/ugebDo9cKGQqm4I3rBZ0FXh/VMdxbH6e/+0snAWdmL36
VuLgXAVHko1hPsHe3PO/DVQhUXQQITMMJ2yUajWCmGHqFIyS9gqVqG9E9WdTSkmxs+2h4g+sk5OuPKdczvzm9Yf5oA49lksQuJcWD3M0MaXnvH07
xwEsQuJiRWdo0JzPXA0OuMcQ1GPUV5E/rMiNn4yjRPP/HAFP7LlfKmkguFfcOsYyXhkNQ2zow9Q4+F12qXiHJGT5ShL4dZWiSU6PCgAmh/cLqFSD
6+ILK4wOBRz9gqlck1pocJJazkP8FaXadW6+pfIWSeVSKQcsZDIXySu453ZsNxAtHOp1/TgtQZFpuarIVSGbUIpwqUacoL3NcuxuBhznHVLUp6WV
vxNks4Z5O4wWH4c3tnE7qrx8r0qcVeuFrTRw96ICkDHqWNEr+gZrIlKAed9KIqGqMzjBZK+QtXDMECCXaS0nIab+ZlRNKFpWiqObLKPkSpLKZ5ow
cuO7EOudaeI6xc50wa7z6FBNMd2oCS9JWt14bbtMLnPXvZ+iMXMgEP929qnFtKZzeRcvkkvnMbaGrqsb/yiQVX5wan6rUzunAWPdTVgcqJT1Pi54
G/OQxiVlcyvg4/PRAfV+8RLW0qeHhJExUVPIS8mz5fE3MIvLNgBHCqsQe/GnLMBV2aUqH1l5o1WsvVTWYJYWZHKZbxpSixxkx1qLeHO+W2NHGJHL
6rWOJctmVuW9IDusIjeGC/L4t1ZygZlkKgpq848PIhMetJxD9j8Aq6GK3gxlXax7dpQ2y/J53kgHbDEvslD5x6MlswhgWcwC9hDcb/gYYTr8BmrZ
d0LtvCzrOJAYsCPObZPqOO37gbykhRhJ2FQv0+Lvp+lj/M5OoRmHtrTPjqNaDVmDncSPTIajXjAItkRxJLJboacSeEsGsJvSD0H0xgUhzhOfK0
QepXXLfzG4aX/ow7we9pOXw3G7ydfdd9iB1yCiIICaW3SAavL2zy/dHMb5/0a0WxMza89pRW8KMZ/GQSxZOS2Ek8fJ954mEbJv8c5ZrzKyC9fbO8
9FsZmHimnBNZBlGyNrKckhBywYcHI/k4ytgkWMpFmYiNxV8j0WVmw1NDXuF/FCnRHHnexgRiVoZU8SWtnBWAqz4gZt3Z9ehoGXYKWXjS8eG0bWX6
ueeNYrNKND5b1zXEd3SlN1UTqrtiqa2NKFAht0DlsMxYqweGTBMk4h06w==''))

# 输出结果（Pass函数产生公钥和Decrypt函数参数太长，不贴出来了，这里这贴下一关的题目，下同）
...
key=input('(x*18-27)/3-(x+7496)=0, x=')
...
```

第二题 (x*18-27)/3-(x+7496)=0, x=?

- 题目提示需要计算(x*18-27)/3-(x+7496)=0, 求x的值。
- 通过简单的数学运算计算出x = 1501，可得到答案。

```
# 第二题
key="1501"

# 输出结果
...
key=input('41*x-31*x^2+74252906=0,(x^2表示x的2次方,下同),x的某个根=')
...
```

第三题 `41*x-31*x^2 + 74252906=0` ,(x^2表示x的2次方,下同),x的某个根=?

- 这一关是求一元二次方程的一个根，通过求根公式可算出。

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

```
# 第三关 解
a = -31
b = 41
c = 74252906
x = (b*(-1) + math.sqrt(b**2 - 4 * a * c) ) / (2 * a)
print(x)
key = '-1547'

# 输出结果
key=input('(1234567^12345678901234567890)%999999997=')
```

第四题 (1234567^12345678901234567890)%999999997=?

- 这一关直接运算会导致溢出，需要用到大数求余的相关知识，具体的自行百度，这里给出代码。

```
# 第四题
a = 1234567
b = 12345678901234567890
c = 999999997
ans = 1
while(b):
    if(b % 2 == 1):
        ans = (ans * a) % c
    b = b // 2
    a = (a * a) % c
print(ans)
key = "42031180"

# 输出结果
key=input('1_2_3_4_5_6_7_8_9=-497,每处_填入1个运算符+-*/,且4个运算符必须都用上,使得等式成立(答案保证唯一),表达式为?')
```

第五题 求表达式

- 题目：1_2_3_4_5_6_7_8_9=-497,每处_填入1个运算符±*/,且4个运算符必须都用上,使得等式成立(答案保证唯一),表达式为?
- 这一关直接通过穷举的方式算出符合条件的表达式，得到4个答案，加上需要满足4个运算符都要用上的条件，可得到唯一的答案。

```
# 第五题 解
expression = ['1', '_', '2', '_', '3', '_', '4', '_', '5', '_', '6', '_', '7', '_', '8', '_', '9']
opertor = "+-*/"
for i1 in range(4):
    expression[1] = opertor[i1]
    for i2 in range(4):
        expression[3] = opertor[i2]
        for i3 in range(4):
            expression[5] = opertor[i3]
            for i4 in range(4):
                expression[7] = opertor[i4]
                for i5 in range(4):
                    expression[9] = opertor[i5]
                    for i6 in range(4):
                        expression[11] = opertor[i6]
                        for i7 in range(4):
                            expression[13] = opertor[i7]
                            for i8 in range(4):
                                expression[15] = opertor[i8]
                                if eval(''.join(expression)) == -497:
                                    print(''.join(expression))
# 打印结果
1+2-3-4+5+6-7*8*9
1-2+3+4-5+6-7*8*9
1-2*3*4+5*6-7*8*9
1/2*3*4-5+6-7*8*9
# 加上限制条件，得到key
key = "1/2*3*4-5+6-7*8*9"

# 输出结果
key=input('x^5-2*x^4+3*x^3-4*x^2-5*x-6=0, x(精确到小数点后14位)=')
```

第六题 `x^5-2*x^4+3*x^3-4*x^2-5*x-6=0`, x(精确到小数点后14位)=?

- 这一关求多次幂的未知数，用到sympy库，具体使用可自行百度官网或其他说明文章。

```
# 第六关 解
x = symbols('x')
key = solve(Eq(x**5-2*x**4+3*x**3-4*x**2-5*x-6, 0),x)
print([N(solution) for solution in key])

key = "2.19488134060852"

# 输出结果
def Hash(context):
 result = md5(bytes(context, 'utf8'))
 for i in range(0, 10000000):
  result = md5(result.digest())
 return result.hexdigest()
key=input('请输入8位数字PIN码:')
print("验证中……")
if Hash(key) == '5f4654140971c47658de19d62ba472b6':
 exec(Decrypt(key,'...')) # 省略密文
else:
 print("PIN码错误")
```

第七题 请输入8位数字PIN码

- 题目要求输入8位PIN码，并进行Hash之后与字符串比较，如果相同则执行解密操作。
- 思路：直接穷举8位数进行解密操作，不进行Hash运算。

```
# 第七题 解
result = 0
for i in range(0, 100000000):
  key = "0"*(8 - len(str(i))) + str(i)
  try:
    Decrypt(key,'rO0WHILNhxyLgMzGsMeRza7URq9BELR/GM/ITqcNWM2IzqUK5J1HCxjC/jU0zgtcgfbgqejdkW+6pxQvdOrQaamGo8N3CMr
vvtBV1mGR5BOjcwoL3gqbOJsiGRl2APzRQ1joKpZUWJuJQEHyriEM9dFMdUStGh2KAx0Q1iFtJ3mioSbY2EbkV7wy7sUQAgkIyxlKYx1gPoJ08p5
HYPQ8McvzCmh4I6SG0TzlFSuuPCPo4C9mPdFeg9V4maHabpwckW8t25ETqo06bsODhJO4kd7mu+evTsuZKjLmiDD3VR4XV1KwKa/Snw2clGwbk48
KTx7iZzTfQi4yDpJtSgDoO6v2zhqSnE5bWupGoRA5xELBIBoQPyHnq/dCxujIlICReJk2HMQKmbuFOvIMF+5Q5eGXoy6M/yAsev9Wc+6WJSoi0fE
YwJKn9bbHVYKjJALGQhw71zaj2Y93DpZa2dT+Cg5bG0QEdzBYpzudi/SPxFmUO1gCMw9gZj/4rUdP7dnzHaAX6pGfbxTRHwY1N4PSXHjjSV40xvu
Pl49i3sAW93i2ibKUTGrMHdmUzhS1lFIrt4ZPIHnXZMLWw/P/Jj4QbZJGfd4BHR6H9drXYkEQLNdBnF0bN6duTID4IJr1Ovle4yANKFDbjd37pS3
1sL0FaazkqB13APKiLrVvpyGAsRC2oGtg/b35zYwrC9Zirj3uGjkYtLjqy6VVZNa2A+WxIFlp6hvA3SXVwog4KdWivzjGtvtMKlniVG6XRIKXuBl
XAJHVBU0TMbU11na32e4aclXRZMisxS2/GU3SlLdaAQdqI+2vEMd8Fkgsh/EbybPUeCX3IlgRqb9gSNjga6DSIyYxFOIUwgtcAS/0TdxvR+ePZH/
G7D0nOALopGRY+xTcUI1QE+oOQKeQEK1dT1cUTcf/EHZUrcIzBO6EcDTu50RQYYAjhRsxrzK3YkFHS0B8/CcRaUyk2ZHLh92q8hvXQewbaDhlKpN
jEAqRzl/E9ppT8IBCWSy1jev5hA1WIb4oY6QvdrE3BgjYbEiVmqXIXcSKhifyhZ4MNI49WpBiC+kyfK7q/RMtU1ogrlDLJTyb2R8GN1NDP7CL43N
5ZJg82eZS2xCGanU0BJSghIiQXcSHCNRE9SsUNnqs/T+NhMuEK/B70kRlnwiNF06qQSUMk1fTOnEa6vY6R50P3FFX30oWvdxvHoynsE3IOyxkfuZ
Six4MmA/LEuFJ613DghXexF6llbo3u7DR+NnUP7nvCS0EHMH7u03qS3FUHQUqsfB+4E3LrP2hBJn68pBY+RKStp0I67TIKbZXiRih8T8ezu+QrME
hVoSsy0F55HvKYdRs6Kj8eWTUraAH0RIEhtBL00H+JtxtmPZXvUkqlA2RQe5m9giN2gMMzTB/DaEtgb4eN9MX+gIghg8E+d1mHnfTm0bIVM628kG
zXSHOu6kGDu40/aapL1boOwfQ5z1mUUjG6k7eJm4iaYbqMMtLzGEVTSTukf2btJURD65RRkZBLahbmUnnn9k84Gg9RZz8TMnVGEZ4QYKo+Gc+6Wv
V38awBTgTjhM1f6Yrg1ecNPvzuMCoLGLpcWMYcl23e2EQXr4YHSQvfoY2iLfegCd9pRrk0Sgi2I5xxYc2Lvq5cdNNZeamK1KNnyxeJaMuOJuX3oA
ddDjT/+ISMupyJCu+I+7OJw4slNIMnbrSoFIrBuTLrkMICHCmUgReer5KQAImQrOpNF2viWLEvbsVWTpWjmYIuUxwQsX27vRg0ZTyBp/yws2fSJK
Mg6Mns/Prnv3EpekpaULPYCy9l5peDmHcebaTHkLeCvCllICw3OSgs1jV1tez3Ccd4vtPC/P7X8Eu9dFunoEpG27utSwWM2Ye/+Zbd2PP/p3QfBZ
2xeSBVihqHGx+6kf0s7rd24Kis3T4PwKkJ4lVay1bHNGUZXpN8KxFILcN/yIfv9ABsd9/QyMRc+0rm1y+QW5ADc2K3OIr5LRLHcCppEixmxW4dgo
pVJwVmkO/v1dNx4kf9A7CG90gK50smJ6556puirf4VXefYTtPAAcBAeJPVDzpNwb5ZzlZKede+hz9fdi3mdniqQX1FpLTYXTMJm0iGYDdOjfPkTO
Q7FICNmPazHAyh5ZN0mgXSBbL+w1dk1zHvFFZFtBkPg/gFOAp3Bn74BjtJtSw9+TGDJ0LfUfKJh5u2hkcG819QXUw1jU4dCsA0ZphoVHk7mzDXc8
iUSQdXDTj3amrkQawVjtGcwagwlJPG9ZrQiaoaKU2g/4koNcPTtpbmu7OzNKZ+J2Ph4i2s4kSuwYKlCi4i/DHBFqcqTOynobIMv0Cj/i0wDD70Rv
LOitHE2rCL4JbI0mLgQmivEcM3etdSuZzmuhQWf9RH/ipvVooE+V9eB9CNV1jU5Okg0NlW8NScz37q7dw0QQQgkmYTAAfDp0ZOhHpuVT/Mc82B2+
IaEEqLCGdXgZ1VCCcUaGjGmo2J1pV6H2X8xK1k+Zk9AZzxjzPWDe8AE1ASTkDo+SzT8Djtq5eKoovj57VybJZpYb7/Wivj++oOvDdm6CL221YMw9
dAoxXlpWdas5QuLbzshiABtog4TmwOHlAjq73qtw9Zmg5g7Wm9vVSENGaD3bxq/bcdDvZLmk9TSiIgBxB2dkb8wDv+0LncqW7QMcN7UBfUeBKcnj
pJ+vU8TF2iGDp3/aD4r4XU1Vr7pQagCy9KAxDu+qw2RlQqWA4cygqqAFNdnxlTdeAu66tv2csORo3vJPG8hhXdM+bJutzewZBDfGZgbFQ39x3PqU
arMXkFeYiipOKLD5zijAIxf2Zg7z2+WWsxkmStLZi0zIBki6T8G7H+9BghwnEs7SZUIwv9iudUTcRBNDvLpN1a6ZAJoGsM4oksq66QdNhCLQIner
dscWpntU3jad5pbSoaUxbxnKvztdRE8LarrE3+5+kgLCZkUBIUZiDvRAn/9KJqujFbCPZLpzK3POfo4Se37/tqxWfm/1fbyOoeFrNZOUOx53PgO9
XJntZ2DrQgcf32V5AkI2oeEWku+jtWwGwK54LCkVWs4FGcI4NChuj+C32oBKwdnQ6VpKSdMUmaLGeNAW0CcP7FAfJB/voR4X93BsZCvLAWVAfCoP
zJ5rEGPYBwhCXGfcN2gvXUK24eYLShgPOOSe5wcseNXAG/kPRuO6Vmhh3vWLBAY4qGkJnjbIrx1/imb3WKy7aQu7f5007EmBSCt70HgBt1+sdhh+
aYPzZPWqJy3ZnAclSAgQIHaId1vCMJS65pIOedo9hZLyFUGkUW7sPoTnXUm9jL/6ZsONP/Kl5KnI9R28sooj7uPzeqyXj5SU2HyqgwRADKL07r8X
6Uudw6a5uD3dxF9B2NLmIWLY+f0YOLEwySDssgNa/0xzLwn1n6fXMW3b0ON+2lsXlzFP/rhAVmR5KE3ef/b24z8TEBNaV+CGbjZpf9d6C8gzuOsi
effKPuQmIEVXdPlnArPYkpUWKhvFZzRvlSF8FdDRWQriWTlGljEDOTR9eEAXgbf2SPc2QFrk3dhto7JYaljKY2scIW596WjYXkbhQqZhLTXy+gzQ
tPtYPtAyyBrHVCAgqyEs6SXH/0nDnzSdd1BHlxZtA/MWuF9aA+XfEK1DwHAJaPEcQ06qlxjF6MQPS7JuS7LVQAHephMnfB8cQtgM1H3vphs5N45j
l5kBY+eQcFhJrKTCOdSXSMkjvrFJfMfld4+btCCkgtLFblSYaYXXPGXKLFZbPOipqrhSmQrH2t7fBKcWgzCRJb8Za3ur0rG4CUOGIVAD1jqs2g2o
ARonBQrt30md4un/NpjTJR8XM0Muk1i3zv7TZu5gBy98Nvw1v')
    print("pin码: ", key)
    result = key
    break
  except:
    if(i % 100000 == 0):
        print("wrong:",key)

key = "79547124"

# 输出结果
stack = []
ins_len = [1] * 5 + [2] * 9 + [9, 1]
reg = [0] * 16
code = base64.b64decode('zyLpMs8CL9Oy/3QDdRlURZRGFHQHdRhURZFGIL/lv+MiNi+70AXRBtMD1wfYCNkJ5v3/iV14RWMB0n+/xgk=')
while True:
```

```
 ins, r0 = code[reg[15]] >> 4, code[reg[15]] & 15
 length = ins_len[ins]
 if length > 1:
  arg = code[reg[15] + 1 : reg[15] + length]
  if length == 2: r1 = arg[0] >> 4; r2 = arg[0] & 15
 reg[15] += length
 if 0 == ins : break
 elif 1 == ins : stack.append(reg[r0])
 elif 2 == ins : reg[r0] = stack.pop()
 elif 3 == ins :
  if not reg[r0] : reg[15] += ins_len[code[reg[15]] >> 4]
 elif 4 == ins : reg[r0] = 0 if reg[r0] else 1
 elif 5 == ins : reg[r0] = reg[r1] + reg[r2]
 elif 6 == ins : reg[r0] = reg[r1] - reg[r2]
 elif 7 == ins : reg[r0] = reg[r1] * reg[r2]
 elif 8 == ins : reg[r0] = reg[r1] / reg[r2]
 elif 9 == ins : reg[r0] = reg[r1] % reg[r2]
 elif 10 == ins : reg[r0] = 1 if reg[r1] < reg[r2] else 0
 elif 11 == ins : stack.append(reg[r0]); reg[r0] += int.from_bytes(arg, byteorder='little', signed=True)
 elif 12 == ins : reg[r0] += int.from_bytes(arg, byteorder='little', signed=True)
 elif ins in (13, 14) : reg[r0] = int.from_bytes(arg, byteorder='little', signed=True)

key = str(reg[0])+str(reg[1])
```

第八题 算出reg[0]和reg[1]

- 通过调试方式分析程序功能为运行表达式多次，结果保存在reg[]中。

```
stack = []    # 模拟栈行为
ins_len = [1] * 5 + [2] * 9 + [9, 1]  # 每个指令的长度
print("ins_len:", ins_len)
reg = [0] * 16  # 模拟寄存器
print("reg", reg)
code = base64.b64decode('zyLpMs8CL9Oy/3QDdRlURZRGFHQHdRhURZFGIL/lv+MiNi+70AXRBtMD1wfYCNkJ5v3/iV14RWMB0n+/xgk=')
print("code:", code)  # 字节格式
hexStr = ""
codeLen = 0
for byte in code:
    hexStr += str(hex(byte))[2:] + " "
    codeLen += 1
print(hexStr)  # 十六进制代码格式
print("codeLength:", codeLen)
lengthList = [0]*16
#stack = [61]
#reg = [5, 6, 4, 3, 0, 0, 99999999999999997, 7, 8, 9, 0, 0, 0, 0, 0, 3]
count = 0
while True:
    # input("step")
    print("code[%d]" %(reg[15]), hex(code[reg[15]]))
    ins, r0 = code[reg[15]] >> 4, code[reg[15]] & 15  # 取字节的高4位作为指令值，低4位作为r0参数
    print("序号:", reg[15], "指令号", ins)
    length = ins_len[ins]     # 获取指令长度
    # 测试代码
    # lengthList[ins] += 1
    print("length=", length)
    # print("ins_len   :", ins_len)
    if length > 1:
        arg = code[reg[15] + 1 : reg[15] + length]
        print("arg[%d:%d]:" %(reg[15] + 1, reg[15] + length), arg)
```

```
        if length == 2:
            r1 = arg[0] >> 4; r2 = arg[0] & 15
            print("r1=", r1, " r2=", r2)    # 指令长度为2的，去第二个字节的高4位和低4位分别作为r1和r2参数

    print("reg[%d]=" % (r0), reg[r0])
    reg[15] += length  # 下一条code的位置
    print("stack=",stack)
    print("reg_before:", reg)
    print("ins, r0:", ins, r0)
    if 0 == ins : break     # 跳出循环，结束
    elif 1 == ins :
        stack.append(reg[r0])   # 加入栈值
        print("ins1: 加入栈值, stack=", stack)
    elif 2 == ins :
        reg[r0] = stack.pop()   # 取出栈值
        if r0 == 0:    # 计算运行次数
            count += 1
        print("ins2: 取出栈值, stack=", stack)
    elif 3 == ins :
        print("ins3 若reg[%d]=%d为零，则reg[15]加上一个数" %(r0, reg[r0]))
        if not reg[r0] :
            reg[15] += ins_len[code[reg[15]] >> 4]      # 赋值运算
            print("ins3 reg[15]+= ins_len[code[reg[15]] >> 4] = " , reg[15])
        print("reg[15]=", reg[15])
    elif 4 == ins :
        print("ins4 reg[%d]=" %(r0), reg[r0])
        reg[r0] = 0 if reg[r0] else 1   # 0 1 互换
    elif 5 == ins :
        reg[r0] = reg[r1] + reg[r2]     # 加
        print("ins5 reg[%d]+reg[%d]=" % (r1, r2), reg[r0])
    elif 6 == ins :
        reg[r0] = reg[r1] - reg[r2]     # 减
        print("ins6 reg[%d]-reg[%d]=" % (r1, r2), reg[r0])
    elif 7 == ins :
        reg[r0] = reg[r1] * reg[r2]     # 乘
        print("ins7 reg[%d]*reg[%d]=" % (r1, r2), reg[r0])
    elif 8 == ins :
        reg[r0] = reg[r1] / reg[r2]     # 除
        print("ins8 reg[%d]/reg[%d]=" % (r1, r2), reg[r0])
    elif 9 == ins :
        reg[r0] = reg[r1] % reg[r2]     # 求余
        print("ins9 reg[%d]%%reg[%d]=" % (r1, r2), reg[r0])
    elif 10 == ins :
        reg[r0] = 1 if reg[r1] < reg[r2] else 0    # 比较赋值
        print("ins10 比较赋值, reg[%d]=" %(r0), reg[r0])
    elif 11 == ins :
        stack.append(reg[r0])       #加入栈值
        temp = int.from_bytes(arg, byteorder='little', signed=True)
        reg[r0] += temp
        print("ins11: 加入栈值,stack:", stack)
        print("ins11 reg[%d]+%d=" %(r0, temp), reg[r0])
    elif 12 == ins :
        temp = int.from_bytes(arg, byteorder='little', signed=True)
        reg[r0] += temp
        print("ins12 reg[%d]+%d=" % (r0, temp), reg[r0])
    elif ins in (13, 14) :
        temp = int.from_bytes(arg, byteorder='little', signed=True)
        reg[r0] = temp
        print("ins13-14 reg[%d]=arg=%d" %(r0, temp))
    print("reg_after:", reg)
```

```
    print("reg_after:", reg)
    print("reg[%d]=" % (r0), reg[r0])
    print("stack=", stack)
    print("count=", count)
    print("----------------------")

print("reg_after:", reg)
print("stack=", stack)
print("count=", count)
```

- 可初步分析出code有以下功能。

```
# 初始化
def init():
    print("初始化函数")
    reg[0] = 5
    reg[1] = 6
    reg[3] = 3
    reg[7] = 7
    reg[8] = 8
    reg[9] = 9
    reg[6] = 99999999999999997
    reg[2] = 127

# 运算reg[0],reg[1]
def operation():
    reg[4] = reg[0] * reg[3]
    reg[5] = reg[1] * reg[9]
    reg[4] = reg[4] + reg[5]
    reg[4] = reg[4] % reg[6]
    temp = reg[4]

    reg[4] = reg[0] * reg[7]
    reg[5] = reg[1] * reg[8]
    reg[4] = reg[4] + reg[5]
    reg[1] = reg[4] % reg[6]
    reg[0] = temp

# 运算次数通过修改reg[2]=127的值进行调试，找到规律，是累加的累加，计算次数的函数如下
def calculateNum(time):
    num = 1
    sum = [0]
    while num <= time:
        temp = 0
        for i in range(num):
            temp += sum[i]
        sum.append(num + temp)
        num += 1
    print("运算次数:", sum[time])
    return sum[time]
```

- 对上述程序运算过程进行优化，得到如下测试代码，用于验证前几次运算的结果是否正确。说明code的运算过程没有问题。

```
# 运算过程
def operation(x, y):
    temp = (3*x + 9*y) % 99999999999999997
    y = (7*x + 8*y) % 99999999999999997
    x = temp
    return x, y
# 计算运算次数
def calculateNum(time):
    num = 1
    sum = [0]
    while num <= time:
        temp = 0
        for i in range(num):
            temp += sum[i]
        sum.append(num + temp)
        num += 1
    print("运算次数:", sum[time])
    return sum[time]
# 测试运算结果
def test(time):
    x = 5
    y = 6
    time = calculateNum(time)
    for i in range(time):
        x, y = operation(x, y)
        print("x=%d, y=%d" % (x, y))
test(4)
```

- 根据reg[2] = 127，calculateNum(127)计算运算次数为`170141183460469231731687303715884105727`。
- 可以发现运算的次数非常大，是不可能快速运算完的，需要优化算法。
- 可通过矩阵点乘的方式算出2^n次运算对应的常数列表，然后根据剩余的运算次数选择相应的常数值进行运算。
- 代码如下。

```
MatrixA = np.array([[3, 9], [7, 8]])
MatrixList = []
MatrixList.append(MatrixA)
# 生成常数矩阵列表
def CreateMatrixList(powerNum):
    MatrixB = np.array([[3, 9], [7, 8]])
    for i in range(powerNum):
        MatrixB00 = int(MatrixB[0][0])
        MatrixB01 = int(MatrixB[0][1])
        MatrixB10 = int(MatrixB[1][0])
        MatrixB11 = int(MatrixB[1][1])
        temp00 = MatrixB00 * MatrixB00 % 99999999999999997 + MatrixB01 * MatrixB10 % 99999999999999997
        temp01 = MatrixB00 * MatrixB01 % 99999999999999997 + MatrixB01 * MatrixB11 % 99999999999999997
        temp10 = MatrixB10 * MatrixB00 % 99999999999999997 + MatrixB11 * MatrixB10 % 99999999999999997
        temp11 = MatrixB10 * MatrixB01 % 99999999999999997 + MatrixB11 * MatrixB11 % 99999999999999997
        MatrixB = np.array([[temp00, temp01], [temp10, temp11]]) % 99999999999999997
        # 计划添加 列表，保存之前计算过的值
        MatrixList.append(MatrixB)

# 矩阵运算
def MatrixOperation(x, y, test):
    a00 = int(test[0][0])
    a01 = int(test[0][1])
    a10 = int(test[1][0])
```

```
    a11 = int(test[1][1])
    temp = (a00*x%9999999999999997 + a01*y%9999999999999997)%9999999999999997
    y = (a10*x%9999999999999997 + a11*y%9999999999999997)%9999999999999997
    x = temp
    return x, y

# 计算reg[0], reg[1]
def CalculateKey(time):
    # 运算次数
    # countContrl = 170141183460469231731687303715884105727
    count = calculateNum(time)
    CreateMatrixList(time)
    x = 5
    y = 6
    while count != 0:
        # 将运算次数求对数
        powerNum = int(math.log(count, 2))
        # 根据运算次数求幂
        num = int(math.pow(2, powerNum))
        print("powerNum:", powerNum)
        # 选取对应的幂值，如果运算次数小于求幂的值，说明取对应的矩阵值会溢出
        if count < num:
            powerNum -= 1
            num = int(math.pow(2, int(powerNum)))
            if count < num:
                print("致命错误！退出")
                break
            print("小于操作")
        elif count > num:
            print("大于操作")
        elif count == 0:
            # 退出
            print("退出")
            break
        else:
            print("等于操作")
        # 运算过程
        count -= num
        x, y = MatrixOperation(x, y, MatrixList[powerNum])

        print("剩余次数:", count)
        print("已运算次数:", num)
        print("幂:", powerNum)
        print("MatrixList[%d]=" % (powerNum), MatrixList[powerNum])
        print("x=%d, y=%d" % (x, y))
        print("----------------------")
        # break

# 计算reg[0], reg[1]
CalculateKey(127)
```

总结

- 总结一下之前解题的思路。
- 熟悉一下writeup文章的编写。