碰碰碰，撞撞撞

赛题描述

Alice总是喜欢给Bob出Crypto赛题玩儿，还不让外人知晓赛题的内容。经过chen_xing的不懈努力，终于发现了Alice与Bob总是采用一台年代久远的密码机通讯，该密码机采用Diffie-Hellman密钥协商协议。与此同时，细心的chen_xing还统计分析了该密码机的Diffie-Hellman协议协商的密钥长度频谱图。今天，Alice又给Bob出题了，chen_xing也想写一写这道题，于是他截获了公开信道的一些数据，但是他发现他根本不会写，现在chen_xing把截获的信息交给你，你能否帮他解出这道题目？
今天的TA公共参数如下：

```
g = 0x163d
p = 0x17fe488340020cf9ac3016a2ec2baaa199d09bd77a7eb7d7920e6269093c62173
```

附件

```
hint.png # 关于密码机协商密钥时使用的临时私钥的长度频谱图
Alice.csv # Alice协商密钥时产生的侧信道计时
Bob.csv # Bob协商密钥时产生的侧信道计时
encrypt.py # 赛题描述中针对于 Alice 给 Bob 出的题目的加密算法 符合Kerckhoffs's principle
final_task.enc # 采用 encrypt.py 加密后的 server.py 的密文
```

赛题解析

- 捕捉赛题描述与附件中的关键信息：

  1. Diffie-Hellman密钥协商协议

     公共参数：g, p
     $Alice$：
     $choose \quad a \in_R [1, p-1]$
     $comput \quad g^a (mod\ p) \rightarrow Bob$
     Bob：
     $choose \quad b \in_R [1, p-1]$
     $comput \quad g^b (mod\ p) \rightarrow Alice$
     $key_{session} = (g^a)^b = (g^b)^a = g^{a \cdot b} (mod\ p)$

  2. **RSA计时攻击**

     $pow(a, b)$/快速幂算法：
     $b = 0b1 \cdots$
     $for \quad i \quad in len(b.bit\_length())$
     $result = result * result$
     $if \quad bit_i = 1 \quad result * a$

     由于在底层**快速幂算法**实现的过程中，**模乘法**在单次循环中使用的次数不同，因此会导致消耗的时间呈现出较大的差异，这就会导致**RSA的侧信道计时攻击**。攻击思路简单的来说就是，看起来比较长的时间就意味着该比特位为**1**，反之为**0**，不过不论是现实中的侧信道攻击，还是在这里的**demo**，都会是存在**噪声**的。因此我们要进行足够样本提取，之后**侧信道攻击**恢复临时私钥**a**、**b**，这也是为何会给大家存有大量的时间数据的样本文件。

  3. hint.png
     该图像是为了提醒选手，我们通过**RSA侧信道攻击**要恢复的私钥比特位长度为**256**，按照预期的思路应该是根据高斯分布的频谱图，逐一尝试长度，按照这个思路，我们首先尝试的也应该是**256-bit**。

- 基本信息就这么多，那这个时候我们就要恢复会话密钥，还原**server.py**文件了，我使用的脚本如下：

```python
import statistics

num_of_samples = '50'
num_of_bits = 255
g = 0x163d
p = 0x17fe488340020cf9ac3016a2ec2baaa199d09bd77a7eb7d7920e6269093c62173

def solve(filename):

    with open(filename) as f:
        lines = f.readlines()

    ts = [int(x.strip()) for x in lines]
    median = statistics.median(ts)
    outlier = median * 3
```

```
    tss = []
    for i in range(num_of_bits):
        tmp = []
        for j in range(int(num_of_samples)):
            value = ts[i + j * num_of_bits]
            if value > outlier:
                tmp.append(median)
            else:
                tmp.append(value)
        tss.append(tmp)
    guessed = ''
    threshold = int(statistics.mean([statistics.mean(x) for x in tss]))
    print('threshold is', threshold)
    for i in range(num_of_bits):
        if statistics.mean(tss[i]) >= threshold:
            guessed += '1'
        else:
            guessed += '0'
    guessed_bin = '1' + guessed[::-1]
    guessed_key = int(guessed_bin, 2)
    print(f"The key of {filename}: {guessed_key}")
    return guessed_key

a = solve('Alice.csv')
b = solve('Bob.csv')
session_key = pow(g, a * b, p)
print(f"session_key = {session_key}")
"""
threshold is 138
The key of Alice.csv: 650149653454271299152746968900864808358828172288686085746111572595188432449 73
threshold is 138
The key of Bob.csv: 946375194308735937065308440167761848339645986466399335655881028486352680459 67
session_key = 886844605413586422670555264589213899438571336462861941702353834699081557151 19
"""
```

这里我取的样本数据量为**50**，这里其实并不是很困难，就是机器学习最基础的一个样本抽取与计算平均数据，以平均数据作为界限划分预测比特，这里在赛前已经测试过了，难度很低，如果我们**样本数量≥7**就可以完成恢复，在生成数据的程序中，已经将噪声降得很低了。

- 恢复**server.py**

```
from Crypto.Cipher import AES
from Crypto.Util.Padding import unpad
import hashlib

session_key = 886844605413586422670555264589213899438571336462861941702353834699081557151 19
with open('final_task.enc', 'rb') as f:
    ciphertext = f.read()
key = hashlib.md5(hex(session_key)[2:].encode()).digest()
cipher = AES.new(key = key, mode = AES.MODE_ECB)
message = unpad(cipher.decrypt(ciphertext), 16).decode()
with open('task.py', 'w') as f:
    f.write(message)
```

恢复得到的**server.py**如下：

```
# -*- coding: utf-8 -*-

from Crypto.Util.number import long_to_bytes, bytes_to_long, getPrime, inverse
from hashlib import md5
from os import urandom
from secret import FLAG

import base64
import random
import threading
import socketserver
import string

class Challenge:
```

```
    def __init__(self):
        self.p = getPrime(1024)
        self.q = getPrime(1024)
        self.n = self.p * self.q
        self.phi = (self.p - 1) * (self.q - 1)
        self.user_db = []

    def save(self, username, keyint):
        key = long_to_bytes(keyint)
        key_commit = md5(key).digest()
        username_int = bytes_to_long(username)
        username_enc = pow(username_int, keyint, self.n)
        self.user_db.append((username_enc, key_commit))

    def check(self, index, keyint):
        key = long_to_bytes(keyint)
        assert md5(key).digest() == self.user_db[index][1], "Invalid key"
        username_enc = self.user_db[index][0]
        d = inverse(keyint, self.phi)
        username_dec = pow(username_enc, d, self.n)
        username = base64.b64encode(long_to_bytes(username_dec)).decode()
        assert "VCTF" in username, "Not Admin"

class Task(socketserver.BaseRequestHandler):

    def __init__(self, *args, **kargs):
        super().__init__(*args, **kargs)
        self.timeout_timer = None

    def timeout_handler(self):
        raise TimeoutError("Connection timed out (600s)")

    def dosend(self, msg):
        try:
            self.request.sendall(msg.encode("utf-8") + b"\n")
        except:
            pass

    def register(self):
        self.dosend(f"Give me your key >>>")
        buf = b""
        while len(buf) < 1000:
            recv_data = self.request.recv(1)
            if not recv_data:
                return
            buf += recv_data
            if buf[-1:] == b"\n":
                break
        your_key = buf.decode().strip()

        try:
            your_key_int = int(your_key, 16)
        except ValueError:
            self.dosend("Error: Key must be hexadecimal (e.g., a1b2c3)")
            return

        self.dosend(f"Give me your username >>>")
        buf = b""
        while len(buf) < 10:
            recv_data = self.request.recv(1)
            if not recv_data:
                return
            buf += recv_data
            if buf[-1:] == b"\n":
                break
        buf = buf[:-1].strip()

        try:
            assert "vctf" in buf, "Not admin."
            assert len(buf) <= 5, "Too long."
```

```
            self.chall.save(buf, your_key_int)
            self.dosend("Register success! Your index is: " + str(len(self.chall.user_db) - 1))
        except AssertionError as e:
            self.dosend(f"Register failed: {e}")
            return

    def get_flag(self):
        self.dosend(f"Give me your key >>>")
        buf = b""
        while len(buf) < 1000:
            recv_data = self.request.recv(1)
            if not recv_data:
                return
            buf += recv_data
            if buf[-1:] == b"\n":
                break
        your_key = buf.decode().strip()

        try:
            your_key_int = int(your_key, 16)
        except ValueError:
            self.dosend("Error: Key must be hexadecimal (e.g., a1b2c3)")
            return

        self.dosend(f"Give me your index >>>")
        buf = b""
        while len(buf) < 1000:
            recv_data = self.request.recv(1)
            if not recv_data:
                return
            buf += recv_data
            if buf[-1:] == b"\n":
                break
        index_str = buf.decode().strip()

        try:
            index = int(index_str)
            assert index >= 0 and index < len(self.chall.user_db), f"Invalid index (must be 0~
{len(self.chall.user_db)-1})"
        except ValueError:
            self.dosend("Error: Index must be an integer (e.g., 0)")
            return
        except AssertionError as e:
            self.dosend(f"Index error: {e}")
            return

        try:
            self.chall.check(index, your_key_int)
            self.dosend(f"🎉FLAG: {FLAG}")
        except AssertionError as e:
            self.dosend(f"Get flag failed: {e}")
            return

    def handle(self):
        try:
            self.timeout_timer = threading.Timer(600, self.timeout_handler)
            self.timeout_timer.start()

            self.chall = Challenge()
            self.dosend(f"p = {self.chall.p}")
            self.dosend(f"q = {self.chall.q}")

            while True:
                self.dosend(f"[R]egister")
                self.dosend(f"[G]et Flag")
                self.dosend(f"[E]xit")
                self.dosend(f"Your choice >>>")

                buf = b""
                while len(buf) < 1000:
```

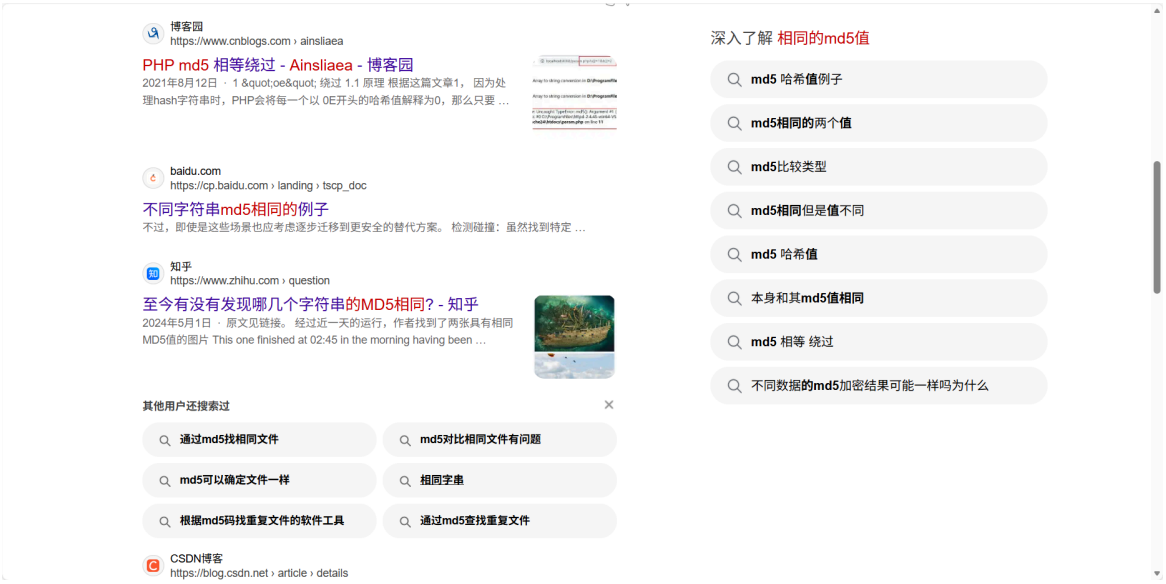
```
                    recv_data = self.request.recv(1)
                    if not recv_data:
                        return
                    buf += recv_data
                    if buf[-1:] == b"\n":
                        break
                choice_str = buf.decode().strip()

                try:
                    choice = choice_str
                except ValueError:
                    self.dosend("Error: Invalid input.")
                    continue

                if choice.upper() == "R":
                    self.register()
                elif choice.upper() == "G":
                    self.get_flag()
                elif choice.upper() == "E":
                    self.dosend("Goodbye!")
                    self.timeout_timer.cancel()
                    self.request.close()
                    return
                else:
                    self.dosend("Error: Invalid input.")
        except TimeoutError as e:
            self.dosend(f"Timeout: {e}")
            return
        except Exception as e:
            return
        finally:
            if self.timeout_timer:
                self.timeout_timer.cancel()
            self.request.close()


class SimpleServer(socketserver.TCPServer):
    pass


if __name__ == "__main__":
    HOST, PORT = "0.0.0.0", 8888
    try:
        server = SimpleServer((HOST, PORT), Task)
        server.allow_reuse_address = True
        server.serve_forever()
    except OSError as e:
        print(f"Error: {e}")
    except KeyboardInterrupt:
        print("Server have died.")
```

- 之后我们就能拿到**server**运行的**server.py**了，分析一下逻辑：
    1. 用户名**username**必须包含**vctf**并且**len(username) ≤ 5**。
    2. 之后就是注册时的密码要求并且计算**bytes_to_long(key)**，存储**md5(key)**用于后期校验，之后以**bytes_to_long(key)**作为加密指数加密**bytes_to_long(username)**，设密文为**ciphertext**。
    3. 当我们在登录时，我们需要输入**username**与**key**，我们在这里设为 **key'**。首先完成的就是**md5(key) == md5(key')**的校验，这也是我们所说的常规登录中的存储真实**key**的**哈希值**的操作。之后，服务器会利用**key'**作为加密指数**e**来**RSA**解密**ciphertext**，最终对于解密结果进行**base64**编码，如若编码后的结果内存在**VCTF**，则登录成功，也就是挑战成功。如果中间存在一个不满足的条件，就算是挑战失败。
- 思考如何完成挑战：
    - 首先我们要满足的就是登录与注册时的**username**与**key**的选择，如果说我们输入的**key' == key**，那么就相当于时正常的RSA加解密，针对于我们输入的**vctf**/**vctf+byte**/**byte+vctf**进行加密，之后采用正确的解密，那得到的就是我们输入的原始的**username**，之后对于原始的**username**进行**base64编码**，此时令编码结果存在**VCTF**几乎是不可能的。
      vctf：这里就不说了，加解密均正确的话，就是对**vctf**编码，之后要求编码结果里存在**VCTF**
      byte+vctf/**vctf+byte**：前者**a3ZjdGY=**，前两位可变，存在**VCTF**概率为**0**；对于后者会是**dmN0Z__=**的结果，后续两位可变，存在**VCTF**概率为**0**
    - 这个时候我们就要考虑，我能否找到一个**key' != key 且 md5(key') == md5(key)**，这样就可以保证**md5**校验成功，并且错误的加解密会导致解密后的内容很长，此时再进行**base64**编码后存在**VCTF**的概率就会大大增加，而我们可以寻找的明文空间为

vctf/vctf+byte/byte+vctf，我们对于一组**(p,q)**，采用**RSA**加密明文空间内的所有内容，再利用**key'**进行解密、编码，查看此时的数据中是否存在**VCTF**。若存在，则进行注册、登录、获取**flag**，反之重新连接靶机更新**(p, q)**。

- 这就是预期的思路，那这个时候还有一个问题没有解决，那就是我该怎么找到**key' != key 且 md5(key') == md5(key)**这样的一组**(key', key)**的组合呢？

1. 我认为密码学本身就是在前人的经验和基础上完成的架构，我们可以查看网上的相关资料进行搜索，例如：[至今有没有发现哪几个字符串的MD5相同? - 知乎]



2. 由于**md5**在当前的环境中并不是安全的，也存在一些工具进行相关的爆破：**fastcoll**.....

- 写交互程序完成挑战获取**flag**，这里就不再连接靶机了，进行一个本地测试的程序放在这里：

```
import multiprocessing
import time
import string
from Crypto.Util.number import *
from hashlib import md5
import base64

keyA_hex = "54455854434f4c4c42596647694a5545544851346841634b534d64357a597067716631595244686b6d78486b6850577074726b6f797a3238776e4939563061486541756146b6e616b"
keyB_hex = "54455854434f4c4c42596647694a5545544851346845634b534d64357a597067716631595244686b6d78486b6850577074726b6f797a3238776e4939563061486541756146b6e616b"
keyA_int = int(keyA_hex, 16)
keyB_int = int(keyB_hex, 16)


call_count = multiprocessing.Value('i', 0)
stop_flag = multiprocessing.Value('b', False)

class Challenge:
    def __init__(self):

        self.p = getPrime(1024)
        self.q = getPrime(1024)
        self.n = self.p * self.q
        self.phi = (self.p - 1) * (self.q - 1)
        self.user_db = []

    def save(self, username, keyint):
        key = long_to_bytes(keyint)
        key_commit = md5(key).digest()
        username_int = bytes_to_long(username)
        username_enc = pow(username_int, keyint, self.n)
        self.user_db.append((username_enc, key_commit))

    def check(self, index, keyint):
        key = long_to_bytes(keyint)
        username_enc = self.user_db[index][0]
```

```
        try:
            d = inverse(keyint, self.phi)
        except:
            return False
        username_dec = pow(username_enc, d, self.n)
        username_bytes = long_to_bytes(username_dec)

        if len(username_bytes) % 3 == 1:
            username_bytes += b'\x00\x00'
        elif len(username_bytes) % 3 == 2:
            username_bytes += b'\x00'

        username_b64 = base64.b64encode(username_bytes).decode()
        return "VCTF" in username_b64

def generate_usernames():
    usernames = [b'vctf']
    high_prob_chars = "AEIMQUYcgkqwz"

    for c in high_prob_chars:
        usernames.append(c.encode() + b'vctf')
        usernames.append(b'vctf' + c.encode())

    for c in string.ascii_letters + string.digits:
        if c not in high_prob_chars:
            usernames.append(c.encode() + b'vctf')
            usernames.append(b'vctf' + c.encode())

    return usernames

def function_A():
    with call_count.get_lock():
        call_count.value += 1
    chall = Challenge()
    usernames = generate_usernames()

    for username in usernames:
        chall.save(username, keyA_int)
        index = len(chall.user_db) - 1

        if chall.check(index, keyB_int):
            print(f"p = {chall.p}")
            print(f"q = {chall.q}")
            print(f"username = {username}")
            username_bytes = long_to_bytes(pow(chall.user_db[index][0], inverse(keyB_int, chall.phi), chall.n))
            username_b64 = base64.b64encode(username_bytes).decode()
            print(f"Success! Call count: {call_count.value}")
            print(f"Base64 string: {username_b64}")
            return True

    return False

def process_task():
    while not stop_flag.value:
        if function_A():
            with stop_flag.get_lock():
                stop_flag.value = True
            break

def start_processes(process_count):
    processes = []
    for _ in range(process_count):
        p = multiprocessing.Process(target=process_task)
        p.start()
        processes.append(p)

    for p in processes:
        p.join()

if __name__ == "__main__":
```

```
    start_time = time.time()
    start_processes(16)

    print(f"Total calls: {call_count.value}")
    print(f"Time: {time.time() - start_time:.2f}s")
```

这里是开辟了多线程寻找符合要求的内容，如果是在比赛环境中，比赛之前也是测试过的，按照我的思路，多次连接靶机，时间足够完成挑战。