**广州大学学生实验报告**

**开课学院及实验室：**网络空间安全学院 **2023年4月11日**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **学院** | 网络空间安全学院 | **年级/专业/班** | 网安211 | **姓名** |  | **学号** |  |
| **实验课程名称** | 密码技术及应用 | | | | | **成绩** |  |
| **实验项目名称** | 维吉尼亚密码破译 | | | | | **指导**  **老师** | 李树栋 |

1. 实验目的

（1）了解维吉尼亚密码的简单历史和数学原理，通过加密解密过程，包括如何使用密钥来生成密文以及如何对密文进行解密操作。

（2）通过该实验可以学习维吉尼亚系统的弱点，并使用频率分析和Kasiski测试来破解维吉尼亚密码。

1. 实验内容

（1）对于维吉尼亚密码，练习使用密钥对明文进行加密生成密文；

明文：ATTACKATDAWN，密钥：LEMONLEMONLE，密文：LXFOPVEFRNHR。

（2）对于维吉尼亚密码，练习使用密钥对密文进行解密得到明文的操作；

例：对于下面是一段经过维吉尼亚加密的密文，请你找出它的密钥并解密出原文。

CHREEVOAHMAERATBIAXXWTNXBEEOPHBSBQMQEQERBWRVXUOAKXAOSXXWEAHBWGJMMQMNKGRFVGXWTRZXWIAKLXFPSKAUTEMNDCMGTSXMXBTUIADNGMGPSRELXNJELXVRVPRTULHDNQWTWDTYGBPHXTFALJHASVBFXNGLLCHRZBWELEKMSJIKNBHWRJGNMGJSGLXFEYPHAGNRBIEQJTAMRVLCRREMNDGLXRRIMGNSNRWCHRQHAEYEVTAQEBBIPEEWEVKAKOEWADREMXMTBHHCHRTKDNVRZCHRCLQOHPWQAIIWXNRMGWOIIFKEE

（3）练习针对维吉尼亚密码，使用Kasiski测试确定密钥长度；

（4）频率分析和Kasiski测试来破解维吉尼亚密码

（5）练习使用拟重合指数法确定密钥内容。

1. 实验过程及结果

第一题：

# 用python逆加密：用维吉尼亚密码加密明文“please keep this message in secret.”

# ,其中使用的密钥为“computer”,试求其密文。

def vigenere\_cipher(text, key, encrypt=True):

    text = text.lower()

    key = key.lower()   # 使用 lower() 方法将明文转换为小写，并对密钥执行相同操作。

    key\_length = len(key)

    key\_as\_int = [ord(i) - ord('a') for i in key]  # 整数列表，密钥中每个字符的移位值

    result = []  # 初始化结果列表为空列表。

    # for 循环用于遍历文本字符串中的每个字符。

    # enumerate() 函数用于跟踪每个字符的索引。

    for i, c in enumerate(text):

        if c.isalpha():

            shift = key\_as\_int[i % key\_length]  # 如果字符是字母，则通过将模运算符 (%) 与 key\_length 变量一起使用，从 key\_as\_int 列表中获取移位值。 如果 encrypt 为 False，则移位值取反。

            if not encrypt:

                shift = -shift

            new\_c = chr(((ord(c) - ord('a') + shift) % 26) + ord('a'))

            result.append(new\_c)  # 新字符是通过将移位值与原始字符的 ASCII 值相加，减去小写字母 'a' 的 ASCII 值，将结果取模 26 并在必要时环绕到字母表的开头，然后 添加字母“a”的 ASCII 值。 此计算是使用 ord() 和 chr() 函数执行的。

        else:

            result.append(c)  # 如果字符不是字母，则按原样附加到结果列表。

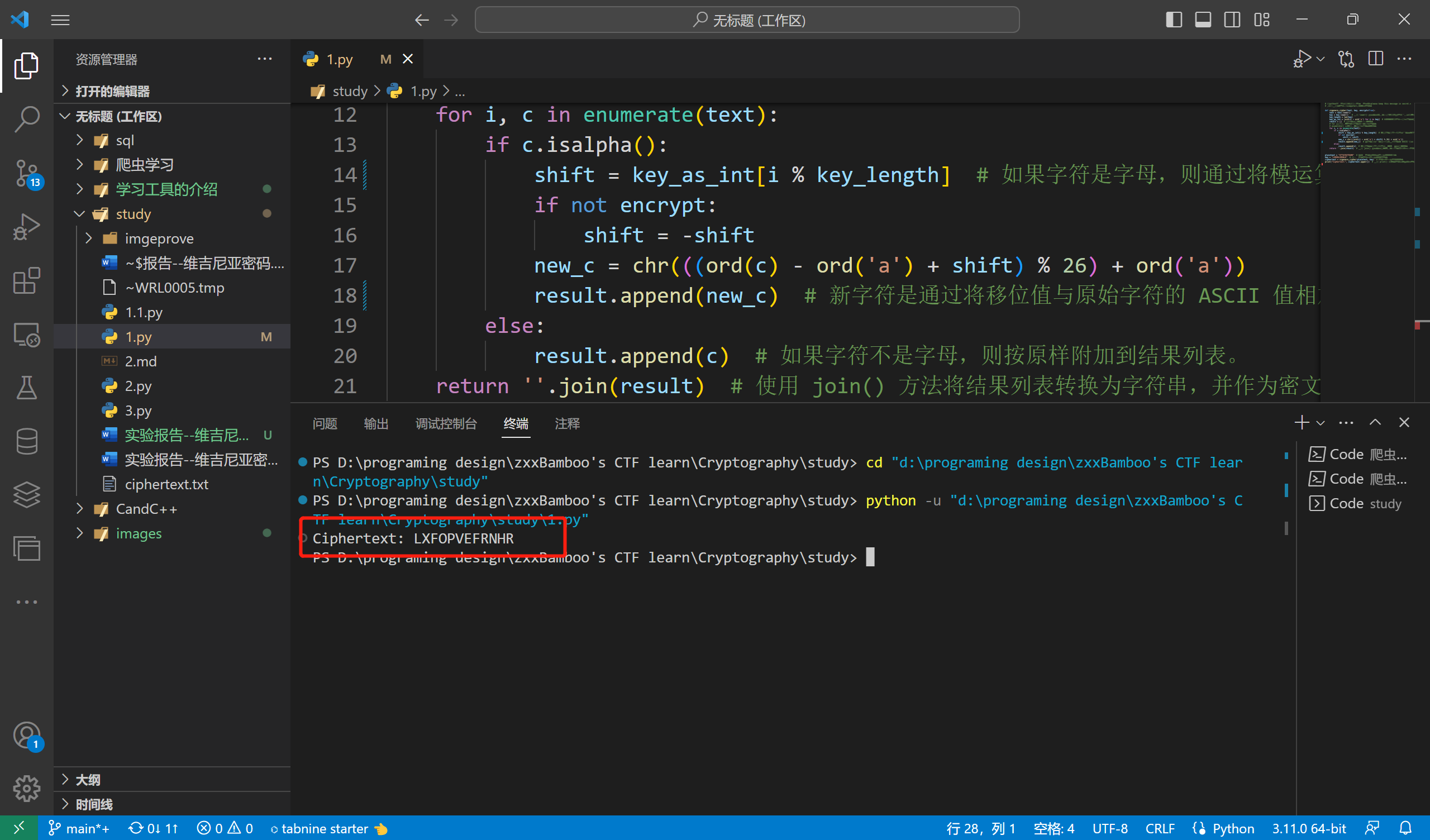
    return ''.join(result)  # 使用 join() 方法将结果列表转换为字符串，并作为密文返回

plaintext = "ATTACKATDAWN"  # 将要加密的明文消息分配给可变文本。

key = "LEMONLEMONLE"  # 用于加密的密钥被分配给变量密钥。

ciphertext = vigenere\_cipher(plaintext, key)  # 密文被分配给密文变量。

print("Ciphertext:", ciphertext.upper())  # 使用 print() 函数将密文打印到控制台。



第二题：

import re

from collections import Counter

from math import gcd

from functools import reduce

def kasiski\_test(ciphertext):

    ciphertext = str(ciphertext)

    # Find repeating sequences of 3 or more characters in the ciphertext

    repeating\_sequences = re.findall(r'(?=(\w{3,})).\*?\1', ciphertext)

    repeating\_sequences = [str(x) for x in repeating\_sequences]

    # Calculate the distance between the repeating sequences

    distances = []

    for seq in set(repeating\_sequences):

        for m in re.finditer(seq, ciphertext):

            distances.append(m.end() - m.start())

    # Determine the greatest common divisor of the distances

    if not distances:

        key\_length = 0

    else:

        key\_length = reduce(gcd, distances)

    return key\_length

def frequency\_analysis(ciphertext, key\_length):

    freqs = 'ETAOINSHRDLCUMWFGYPBVKJXQZ'

    decrypted\_key = ''

    for i in range(key\_length):

        # Group the ciphertext by key length

        segment = ciphertext[i::key\_length]

        # Count the occurrences of each letter in the segment

        letter\_freq = Counter(segment)

        # Find the most common letter in the segment

        most\_common = letter\_freq.most\_common(1)[0][0]

        # Estimate the key character by comparing the most common letter with English letter frequencies

        key\_char = chr(((ord(most\_common) - ord('A')) - (ord(freqs[0]) - ord('A'))) % 26 + ord('A'))

        decrypted\_key += key\_char

    return decrypted\_key

def vigenere\_cipher(ciphertext, key, encrypt=True):

    key\_len = len(key)

    key\_ints = [ord(i) for i in key]

    ciphertext\_ints = [ord(i) for i in ciphertext]

    result = ''

    for i, c in enumerate(ciphertext\_ints):

        key\_int = key\_ints[i % key\_len]

        if encrypt:

            new\_c = (c + key\_int) % 26

        else:

            new\_c = (c - key\_int) % 26

        result += chr(new\_c + ord('A'))

    return result

with open('ciphertext.txt', 'rb') as f:

    ciphertext = f.read()

    ciphertext = ciphertext.decode('utf-8')

# 1. Use the Kasiski test to determine the key length

key\_length = kasiski\_test(ciphertext)

print("Key length:", key\_length)

# 2. Use frequency analysis and the Kasiski test to crack the Vigenère cipher

# 3. Decrypt the ciphertext using the found key

if key\_length == 0:

    print("Failed to determine key length")

else:

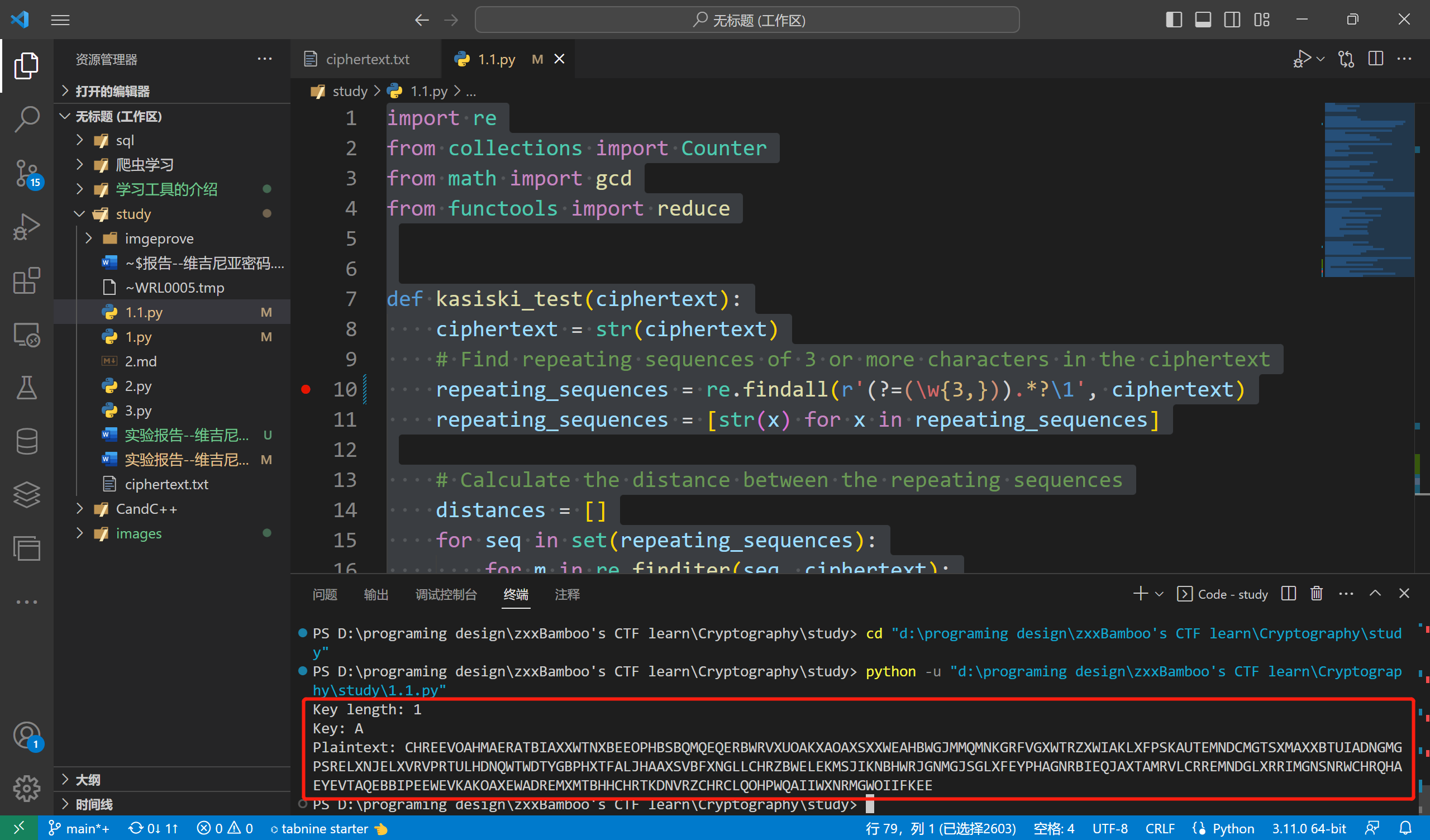
    key = frequency\_analysis(ciphertext, key\_length)

    print("Key:", key)

    plaintext = vigenere\_cipher(ciphertext, key, encrypt=False)

    print("Plaintext:", plaintext)

答案没得出来，错了，因为重复的不止CHR还有一个ELX



换了其他方法，频率统计：

def brute\_force\_vigenere(ciphertext):

possible\_keys = [chr(i) for i in range(ord('A'), ord('Z') + 1)]

for key in possible\_keys:

plaintext = vigenere\_cipher(ciphertext, key, encrypt=False)

print(f"Key: {key}\nPlaintext: {plaintext}\n")

# 在代码的最后调用这个函数

brute\_force\_vigenere(ciphertext)

没用：

Key: B

Plaintext: BGQDDUNZGLZDQZSAHZWWVSMWADDNOGARAPLPDPDQAVQUWTNZJWZNZWRWWVDZGAVFILLPLMJFQEUFWVSQYWVHZJKWEORJZTSDLMCBLFSRWLZWWASTHZCMFLFORQDKWMIDKWUQUOQSTKGCMPVSVCSXFAOGWSEZKIGZZWRUAEWMFKKBGQYAVDKDJLRIHJMAGVQIFMLFIRFKWEDXOGZFMQAHDPIZWSZLQUKBQQDLMCFKWQQHLFMRMQVBGQPGZDXDUSZPDAAHODDVDUJZJNZWDVZCQDLWLSAGGBGQSJCMUQYBGQBKPNGOVPZHHVWMQLFVNHHEJDD

Key: C

Plaintext: AFPCCTMYFKYCPYRZGYVVURLVZCCMNFZQZOKOCOCPZUPTVSMYIVYMYVQVVUCYFZUEHKKOKLIEPDTEVURPXVUGYIJVDNQIYSRCKLBAKERQVKYVVZRSGYBLEKENQPCJVLHCJVTPTNPRSJFBLOURUBRWEZNFVRDYJHFYYVQTZDVLEJJAFPXZUCJCIKQHGILZFUPHELKEHQEJVDCWNFYELPZGCOHYVRYKPTJAPPCKLBEJVPPGKELQLPUAFPOFYCWCTRYOCZZGNCCUCTIYIMYVCUYBPCKVKRZFFAFPRIBLTPXAFPAJOMFNUOYGGUVLPKEUMGGDICC

Key: D

Plaintext: ZEOBBSLXEJXBOXQYFXUUTQKUYBBLMEYPYNJNBNBOYTOSURLXHUXLXUPUUTBXEYTDGJJNJKHDOCSDUTQOWUTFXHIUCMPHXRQBJKAZJDQPUJXUUYQRFXAKDJDMPOBIUKGBIUSOSMOQRIEAKNTQTAQVDYMEUQCXIGEXXUPSYCUKDIIZEOWYTBIBHJPGFHKYETOGDKJDGPDIUCBVMEXDKOYFBNGXUQXJOSIZOOBJKADIUOOFJDKPKOTZEONEXBVBSQXNBYYFMBBTBSHXHLXUBTXAOBJUJQYEEZEOQHAKSOWZEOZINLEMTNXFFTUKOJDTLFFCHBB

Key: E

Plaintext: YDNAARKWDIWANWPXEWTTSPJTXAAKLDXOXMIMAMANXSNRTQKWGTWKWTOTTSAWDXSCFIIMIJGCNBRCTSPNVTSEWGHTBLOGWQPAIJZYICPOTIWTTXPQEWZJCICLONAHTJFAHTRNRLNPQHDZJMSPSZPUCXLDTPBWHFDWWTORXBTJCHHYDNVXSAHAGIOFEGJXDSNFCJICFOCHTBAULDWCJNXEAMFWTPWINRHYNNAIJZCHTNNEICJOJNSYDNMDWAUARPWMAXXELAASARGWGKWTASWZNAITIPXDDYDNPGZJRNVYDNYHMKDLSMWEESTJNICSKEEBGAA

Key: F

Plaintext: XCMZZQJVCHVZMVOWDVSSROISWZZJKCWNWLHLZLZMWRMQSPJVFSVJVSNSSRZVCWRBEHHLHIFBMAQBSROMUSRDVFGSAKNFVPOZHIYXHBONSHVSSWOPDVYIBHBKNMZGSIEZGSQMQKMOPGCYILRORYOTBWKCSOAVGECVVSNQWASIBGGXCMUWRZGZFHNEDFIWCRMEBIHBENBGSAZTKCVBIMWDZLEVSOVHMQGXMMZHIYBGSMMDHBINIMRXCMLCVZTZQOVLZWWDKZZRZQFVFJVSZRVYMZHSHOWCCXCMOFYIQMUXCMXGLJCKRLVDDRSIMHBRJDDAFZZ

Key: G

Plaintext: WBLYYPIUBGUYLUNVCURRQNHRVYYIJBVMVKGKYKYLVQLPROIUERUIURMRRQYUBVQADGGKGHEALZPARQNLTRQCUEFRZJMEUONYGHXWGANMRGURRVNOCUXHAGAJMLYFRHDYFRPLPJLNOFBXHKQNQXNSAVJBRNZUFDBUURMPVZRHAFFWBLTVQYFYEGMDCEHVBQLDAHGADMAFRZYSJBUAHLVCYKDURNUGLPFWLLYGHXAFRLLCGAHMHLQWBLKBUYSYPNUKYVVCJYYQYPEUEIURYQUXLYGRGNVBBWBLNEXHPLTWBLWFKIBJQKUCCQRHLGAQICCZEYY

Key: H

Plaintext: VAKXXOHTAFTXKTMUBTQQPMGQUXXHIAULUJFJXJXKUPKOQNHTDQTHTQLQQPXTAUPZCFFJFGDZKYOZQPMKSQPBTDEQYILDTNMXFGWVFZMLQFTQQUMNBTWGZFZILKXEQGCXEQOKOIKMNEAWGJPMPWMRZUIAQMYTECATTQLOUYQGZEEVAKSUPXEXDFLCBDGUAPKCZGFZCLZEQYXRIATZGKUBXJCTQMTFKOEVKKXFGWZEQKKBFZGLGKPVAKJATXRXOMTJXUUBIXXPXODTDHTQXPTWKXFQFMUAAVAKMDWGOKSVAKVEJHAIPJTBBPQGKFZPHBBYDXX

Key: I

Plaintext: UZJWWNGSZESWJSLTASPPOLFPTWWGHZTKTIEIWIWJTOJNPMGSCPSGSPKPPOWSZTOYBEEIEFCYJXNYPOLJRPOASCDPXHKCSMLWEFVUEYLKPESPPTLMASVFYEYHKJWDPFBWDPNJNHJLMDZVFIOLOVLQYTHZPLXSDBZSSPKNTXPFYDDUZJRTOWDWCEKBACFTZOJBYFEYBKYDPXWQHZSYFJTAWIBSPLSEJNDUJJWEFVYDPJJAEYFKFJOUZJIZSWQWNLSIWTTAHWWOWNCSCGSPWOSVJWEPELTZZUZJLCVFNJRUZJUDIGZHOISAAOPFJEYOGAAXCWW

Key: J

Plaintext: TYIVVMFRYDRVIRKSZROONKEOSVVFGYSJSHDHVHVISNIMOLFRBORFROJOONVRYSNXADDHDEBXIWMXONKIQONZRBCOWGJBRLKVDEUTDXKJODROOSKLZRUEXDXGJIVCOEAVCOMIMGIKLCYUEHNKNUKPXSGYOKWRCAYRROJMSWOEXCCTYIQSNVCVBDJAZBESYNIAXEDXAJXCOWVPGYRXEISZVHAROKRDIMCTIIVDEUXCOIIZDXEJEINTYIHYRVPVMKRHVSSZGVVNVMBRBFROVNRUIVDODKSYYTYIKBUEMIQTYITCHFYGNHRZZNOEIDXNFZZWBVV

Key: K

Plaintext: SXHUULEQXCQUHQJRYQNNMJDNRUUEFXRIRGCGUGUHRMHLNKEQANQEQNINNMUQXRMWZCCGCDAWHVLWNMJHPNMYQABNVFIAQKJUCDTSCWJINCQNNRJKYQTDWCWFIHUBNDZUBNLHLFHJKBXTDGMJMTJOWRFXNJVQBZXQQNILRVNDWBBSXHPRMUBUACIZYADRXMHZWDCWZIWBNVUOFXQWDHRYUGZQNJQCHLBSHHUCDTWBNHHYCWDIDHMSXHGXQUOULJQGURRYFUUMULAQAEQNUMQTHUCNCJRXXSXHJATDLHPSXHSBGEXFMGQYYMNDHCWMEYYVAUU

Key: L

Plaintext: RWGTTKDPWBPTGPIQXPMMLICMQTTDEWQHQFBFTFTGQLGKMJDPZMPDPMHMMLTPWQLVYBBFBCZVGUKVMLIGOMLXPZAMUEHZPJITBCSRBVIHMBPMMQIJXPSCVBVEHGTAMCYTAMKGKEGIJAWSCFLILSINVQEWMIUPAYWPPMHKQUMCVAARWGOQLTATZBHYXZCQWLGYVCBVYHVAMUTNEWPVCGQXTFYPMIPBGKARGGTBCSVAMGGXBVCHCGLRWGFWPTNTKIPFTQQXETTLTKZPZDPMTLPSGTBMBIQWWRWGIZSCKGORWGRAFDWELFPXXLMCGBVLDXXUZTT

Key: M

Plaintext: QVFSSJCOVAOSFOHPWOLLKHBLPSSCDVPGPEAESESFPKFJLICOYLOCOLGLLKSOVPKUXAAEABYUFTJULKHFNLKWOYZLTDGYOIHSABRQAUHGLAOLLPHIWORBUAUDGFSZLBXSZLJFJDFHIZVRBEKHKRHMUPDVLHTOZXVOOLGJPTLBUZZQVFNPKSZSYAGXWYBPVKFXUBAUXGUZLTSMDVOUBFPWSEXOLHOAFJZQFFSABRUZLFFWAUBGBFKQVFEVOSMSJHOESPPWDSSKSJYOYCOLSKORFSALAHPVVQVFHYRBJFNQVFQZECVDKEOWWKLBFAUKCWWTYSS

Key: N

Plaintext: PUERRIBNUZNRENGOVNKKJGAKORRBCUOFODZDRDREOJEIKHBNXKNBNKFKKJRNUOJTWZZDZAXTESITKJGEMKJVNXYKSCFXNHGRZAQPZTGFKZNKKOGHVNQATZTCFERYKAWRYKIEICEGHYUQADJGJQGLTOCUKGSNYWUNNKFIOSKATYYPUEMOJRYRXZFWVXAOUJEWTAZTWFTYKSRLCUNTAEOVRDWNKGNZEIYPEERZAQTYKEEVZTAFAEJPUEDUNRLRIGNDROOVCRRJRIXNXBNKRJNQERZKZGOUUPUEGXQAIEMPUEPYDBUCJDNVVJKAEZTJBVVSXRR

Key: O

Plaintext: OTDQQHAMTYMQDMFNUMJJIFZJNQQABTNENCYCQCQDNIDHJGAMWJMAMJEJJIQMTNISVYYCYZWSDRHSJIFDLJIUMWXJRBEWMGFQYZPOYSFEJYMJJNFGUMPZSYSBEDQXJZVQXJHDHBDFGXTPZCIFIPFKSNBTJFRMXVTMMJEHNRJZSXXOTDLNIQXQWYEVUWZNTIDVSZYSVESXJRQKBTMSZDNUQCVMJFMYDHXODDQYZPSXJDDUYSZEZDIOTDCTMQKQHFMCQNNUBQQIQHWMWAMJQIMPDQYJYFNTTOTDFWPZHDLOTDOXCATBICMUUIJZDYSIAUURWQQ

Key: P

Plaintext: NSCPPGZLSXLPCLEMTLIIHEYIMPPZASMDMBXBPBPCMHCGIFZLVILZLIDIIHPLSMHRUXXBXYVRCQGRIHECKIHTLVWIQADVLFEPXYONXREDIXLIIMEFTLOYRXRADCPWIYUPWIGCGACEFWSOYBHEHOEJRMASIEQLWUSLLIDGMQIYRWWNSCKMHPWPVXDUTVYMSHCURYXRUDRWIQPJASLRYCMTPBULIELXCGWNCCPXYORWICCTXRYDYCHNSCBSLPJPGELBPMMTAPPHPGVLVZLIPHLOCPXIXEMSSNSCEVOYGCKNSCNWBZSAHBLTTHIYCXRHZTTQVPP

Key: Q

Plaintext: MRBOOFYKRWKOBKDLSKHHGDXHLOOYZRLCLAWAOAOBLGBFHEYKUHKYKHCHHGOKRLGQTWWAWXUQBPFQHGDBJHGSKUVHPZCUKEDOWXNMWQDCHWKHHLDESKNXQWQZCBOVHXTOVHFBFZBDEVRNXAGDGNDIQLZRHDPKVTRKKHCFLPHXQVVMRBJLGOVOUWCTSUXLRGBTQXWQTCQVHPOIZRKQXBLSOATKHDKWBFVMBBOWXNQVHBBSWQXCXBGMRBARKOIOFDKAOLLSZOOGOFUKUYKHOGKNBOWHWDLRRMRBDUNXFBJMRBMVAYRZGAKSSGHXBWQGYSSPUOO

Key: R

Plaintext: LQANNEXJQVJNAJCKRJGGFCWGKNNXYQKBKZVZNZNAKFAEGDXJTGJXJGBGGFNJQKFPSVVZVWTPAOEPGFCAIGFRJTUGOYBTJDCNVWMLVPCBGVJGGKCDRJMWPVPYBANUGWSNUGEAEYACDUQMWZFCFMCHPKYQGCOJUSQJJGBEKOGWPUULQAIKFNUNTVBSRTWKQFASPWVPSBPUGONHYQJPWAKRNZSJGCJVAEULAANVWMPUGAARVPWBWAFLQAZQJNHNECJZNKKRYNNFNETJTXJGNFJMANVGVCKQQLQACTMWEAILQALUZXQYFZJRRFGWAVPFXRROTNN

Key: S

Plaintext: KPZMMDWIPUIMZIBJQIFFEBVFJMMWXPJAJYUYMYMZJEZDFCWISFIWIFAFFEMIPJEORUUYUVSOZNDOFEBZHFEQISTFNXASICBMUVLKUOBAFUIFFJBCQILVOUOXAZMTFVRMTFDZDXZBCTPLVYEBELBGOJXPFBNITRPIIFADJNFVOTTKPZHJEMTMSUARQSVJPEZROVUORAOTFNMGXPIOVZJQMYRIFBIUZDTKZZMUVLOTFZZQUOVAVZEKPZYPIMGMDBIYMJJQXMMEMDSISWIFMEILZMUFUBJPPKPZBSLVDZHKPZKTYWPXEYIQQEFVZUOEWQQNSMM

Key: T

Plaintext: JOYLLCVHOTHLYHAIPHEEDAUEILLVWOIZIXTXLXLYIDYCEBVHREHVHEZEEDLHOIDNQTTXTURNYMCNEDAYGEDPHRSEMWZRHBALTUKJTNAZETHEEIABPHKUNTNWZYLSEUQLSECYCWYABSOKUXDADKAFNIWOEAMHSQOHHEZCIMEUNSSJOYGIDLSLRTZQPRUIODYQNUTNQZNSEMLFWOHNUYIPLXQHEAHTYCSJYYLTUKNSEYYPTNUZUYDJOYXOHLFLCAHXLIIPWLLDLCRHRVHELDHKYLTETAIOOJOYARKUCYGJOYJSXVOWDXHPPDEUYTNDVPPMRLL

Key: U

Plaintext: INXKKBUGNSGKXGZHOGDDCZTDHKKUVNHYHWSWKWKXHCXBDAUGQDGUGDYDDCKGNHCMPSSWSTQMXLBMDCZXFDCOGQRDLVYQGAZKSTJISMZYDSGDDHZAOGJTMSMVYXKRDTPKRDBXBVXZARNJTWCZCJZEMHVNDZLGRPNGGDYBHLDTMRRINXFHCKRKQSYPOQTHNCXPMTSMPYMRDLKEVNGMTXHOKWPGDZGSXBRIXXKSTJMRDXXOSMTYTXCINXWNGKEKBZGWKHHOVKKCKBQGQUGDKCGJXKSDSZHNNINXZQJTBXFINXIRWUNVCWGOOCDTXSMCUOOLQKK

Key: V

Plaintext: HMWJJATFMRFJWFYGNFCCBYSCGJJTUMGXGVRVJVJWGBWACZTFPCFTFCXCCBJFMGBLORRVRSPLWKALCBYWECBNFPQCKUXPFZYJRSIHRLYXCRFCCGYZNFISLRLUXWJQCSOJQCAWAUWYZQMISVBYBIYDLGUMCYKFQOMFFCXAGKCSLQQHMWEGBJQJPRXONPSGMBWOLSRLOXLQCKJDUMFLSWGNJVOFCYFRWAQHWWJRSILQCWWNRLSXSWBHMWVMFJDJAYFVJGGNUJJBJAPFPTFCJBFIWJRCRYGMMHMWYPISAWEHMWHQVTMUBVFNNBCSWRLBTNNKPJJ

Plaintext: EJTGGXQCJOCGTCVDKCZZYVPZDGGQRJDUDSOSGSGTDYTXZWQCMZCQCZUZZYGCJDYILOOSOPMITHXIZYVTBZYKCMNZHRUMCWVGOPFEOIVUZOCZZDVWKCFPIOIRUTGNZPLGNZXTXRTVWNJFPSYVYFVAIDRJZVHCNLJCCZUXDHZPINNEJTBDYGNGMOULKMPDJYTLIPOILUINZHGARJCIPTDKGSLCZVCOTXNETTGOPFINZTTKOIPUPTYEJTSJCGAGXVCSGDDKRGGYGXMCMQCZGYCFTGOZOVDJJEJTVMFPXTBEJTENSQJRYSCKKYZPTOIYQKKHMGG

Key: Z

Plaintext: DISFFWPBINBFSBUCJBYYXUOYCFFPQICTCRNRFRFSCXSWYVPBLYBPBYTYYXFBICXHKNNRNOLHSGWHYXUSAYXJBLMYGQTLBVUFNOEDNHUTYNBYYCUVJBEOHNHQTSFMYOKFMYWSWQSUVMIEORXUXEUZHCQIYUGBMKIBBYTWCGYOHMMDISACXFMFLNTKJLOCIXSKHONHKTHMYGFZQIBHOSCJFRKBYUBNSWMDSSFNOEHMYSSJNHOTOSXDISRIBFZFWUBRFCCJQFFXFWLBLPBYFXBESFNYNUCIIDISULEOWSADISDMRPIQXRBJJXYOSNHXPJJGLFF

import re

from collections import Counter

from math import gcd

from functools import reduce

def kasiski\_test(ciphertext):

    ciphertext = str(ciphertext)

    # Find repeating sequences of 3 or more characters in the ciphertext

    repeating\_sequence = "CHR"

    count = ciphertext.count(repeating\_sequence)

    print("Occurrences of 'CHR':", count)

    # Calculate the distance between the repeating sequences

    indices = [m.start() for m in re.finditer(repeating\_sequence, ciphertext)]

    # Calculate the distances between the indices

    distances = [indices[i + 1] - indices[i] for i in range(len(indices) - 1)]

    print('distances:', distances)

    # Determine the greatest common divisor of the distances

    if not distances:

        key\_length = 0

    else:

        key\_length = reduce(gcd, distances)

    return key\_length

def frequency\_analysis(ciphertext, key\_length):

    freqs = 'ETAOINSHRDLCUMWFGYPBVKJXQZ'

    decrypted\_key = ''

    for i in range(key\_length):

        # Group the ciphertext by key length

        segment = ciphertext[i::key\_length]

        # Count the occurrences of each letter in the segment

        letter\_freq = Counter(segment)

        # Find the most common letter in the segment

        most\_common = letter\_freq.most\_common(1)[0][0]

        # Estimate the key character by comparing the most common letter with English letter frequencies

        key\_char = chr(((ord(most\_common) - ord('A')) - (ord(freqs[0]) - ord('A'))) % 26 + ord('A'))

        decrypted\_key += key\_char

    return decrypted\_key

def vigenere\_cipher(ciphertext, key, encrypt=True):

    key\_len = len(key)

    key\_ints = [ord(i) for i in key]

    ciphertext\_ints = [ord(i) for i in ciphertext]

    result = ''

    for i, c in enumerate(ciphertext\_ints):

        key\_int = key\_ints[i % key\_len]

        if encrypt:

            new\_c = (c + key\_int) % 26

        else:

            new\_c = (c - key\_int) % 26

        result += chr(new\_c + ord('A'))

    return result

with open('ciphertext.txt', 'rb') as f:

    ciphertext = f.read()

    ciphertext = ciphertext.decode('utf-8')

# 1. Use the Kasiski test to determine the key length

key\_length = kasiski\_test(ciphertext)

print("Key length:", key\_length)

# 2. Use frequency analysis and the Kasiski test to crack the Vigenère cipher

# 3. Decrypt the ciphertext using the found key

if key\_length == 0:

    print("Failed to determine key length")

else:

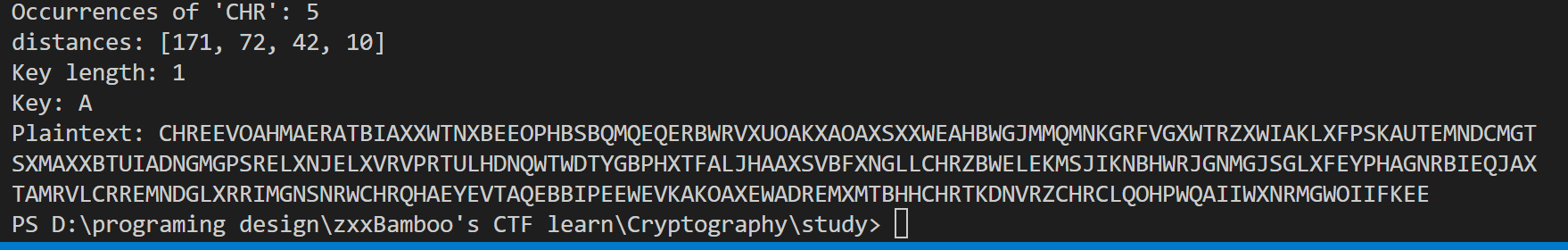
    key = frequency\_analysis(ciphertext, key\_length)

    print("Key:", key)

    plaintext = vigenere\_cipher(ciphertext, key, encrypt=False)

    print("Plaintext:", plaintext)

Kasiski测试没测出来，因为距离间的最大公约数为1。这个显然不对



1. 实验总结