دانشکده مهندسی کامپیوتر

تمرین اول

امنیت سیستم های کامپیوتری

استاد درس: جناب دکتر دیانت

نیم‌سال دوم

سال تحصیلی ۱۴۰۰-۱۴۰۱

فاز ۱

پروژه اول

پارسا عیسی زاده، غزل زمانی نژاد ، آرمان حیدری - ۱۴۰۰/۱۲/۲۷

۱.۱ مقدمه

در زمینه امنیت شبکه بحث رمزنگاری بحثی قابل توجه است . در این پروژه ما قصد داریم یک cipher text را رمزگشایی کنیم . در صورت نداشتن کلید، ساده ترین راه brute force است اما به دلیل زمانگیر بودن این فرآیند باید از راه های دیگری پیاده سازی کنیم و همچنین بعضی از حالات را هم طبق محاسبات و شرایط مساله نادیده بگیریم و بررسی نکنیم .

در کلاس درس ، تضمین شد که رمزنگاری از نوع رمز نگاری ساده تک حرفی است . از این رو ما تنها رمزگشایی های تک حرفی همانند سزار و مستوی را امتحان کردیم .

در cipher-text از آنجایی که همه کلمات ۵ حرفی بودند یک pre-process انجام دادیم و قبل از هر کاری فاصله ها را پاک کردیم .

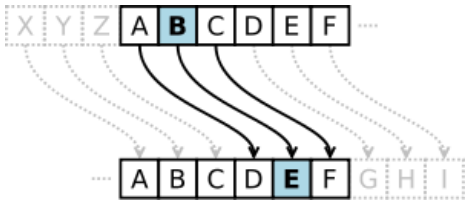
```
۱ def preprocess_cipher(cipher_text):
۲     return cipher_text.replace(' ', '').replace('\n', '')
```

نمونه کد ۱: پیش پردازش برای حذف فاصله ها

۲.۱ روش های به کار گرفته شده

۱.۲.۱ سزار

تعریف سزار : یک نوع رمز جانشینی است که در آن هر حرف در متن آشکار با حرف دیگری با فاصلهٔ ثابت در الفبا جایگزین می‌شود.



اولین و ساده ترین ایده ای بود که بررسی شد و به این صورت بود که در یک حلقه for برای حروف انگلیسی کلید های مختلف را امتحان می کنیم و به plain text می رسیم . سپس هر plain text را بررسی میکنیم که آیا معنی دار است یا خیر.

بررسی معنی دار بودن متن به خاطر زمان بر بودن به شکل دستی انجام نمی شد و ما برای اینکار از پکیج word که در پکیجی به نام nltk بود استفاده کردیم. [۱] در بخش توابع، متد find-words توضیح داده شده است.

در این روش یک کلاس تعریف کردیم به نام DecryptText که علاوه بر متد سازنده سه متد دیگر داشت با نام های :

- is-plain
- find-plain-with-key
- decrypt

در ابتدا متد decrypt روی کل متن فراخوانی می شود در این متد یک brute force زده می شود روی ۲۶ حالت کلید . سپس به ازای هر کلید ، خواستیم plain text استخراج بشود . این استخراج متن توسط متد find-plain-with-key انجام می شود که یک رشته برمیگرداند . سپس این رشته را توسط متد is-plain اعتبار سنجی می کنیم . اگر رشته به دست آمده معتبر بود آن را باز می گردانیم .

```
def decrypt(self):
        for test_key in range(26):
            self.plain = self.find_plain_with_key(test_key)
            if self.is_plain():
                self.key = test_key
                return self.plain
        return text!" plain find t'"Couldn
```

نمونه کد ۲: تابع پیدا کردن کلید های مختلف و شروع عملیات

در متد find-plain-with-key ایندکس هر کاراکتر را برداشتیم و با کلید جمع کردیم و باقی مانده اش را بر ۲۶ گرفتیم تا به کاراکتر رمز شده برسیم سپس کاراکتر های به دست آمده را به متغیر plain text اضافه کردیم

```
def find_plain_with_key(self, key):
        res = ''
        for c in self.cipher:
            new_index = (self.uppercase_list.index(c) + key) % 26
            res += self.uppercase_list[new_index]
        return res
```

نمونه کد ۳: تابع پیدا کردن متن بر اساس کلید

در متد is-plain هر کلمه را درون آرایه ای از کلمات که قبلا داشتیم سرچ میکنیم و اگر وجود داشت به عدد true-records یکی اضافه میکنیم . در نهایت اگر حاصل تقسیم true-record بر تعداد کلمات عدد معقولی بود (از یک نرخ مشخصی بیشتر بود) آن متن را معتبر در نظر میگیریم .

```
def is_plain(self, founded_word_rate=8.0):
        true_records = 0
        words_list = self.plain.split(' ')
        for w in words_list:
            if is_english_word(w):
                true_records += 1
        if true_records / len(words_list) > founded_word_rate:
            return True
        return False
```

نمونه کد ۴: تابع بررسی اعتبار متن

امتحان کردن این روش زمان زیادی نگرفت و ایده ای بسیار ساده بود . این روش پاسخگو نبود.

۲.۲.۱ تحلیل فرکانسی

در این رمز نگاری هر حرف به یک حرف دیگر map می شود . فرقش با سزار این است که لزوما ترتیب حروفی که به آنها map می شوند می تواند یکی نباشد با ترتیب حروفی که map می شوند.

به عبارتی هر حرف می تواند به حرفی رندوم map شود. یکی از راه هایی که برای شکستن این کد کاربرد دارد این است که از تعداد نسبی رخداد کلمات استفاده کنیم.

از انجایی که حروف رندوم و بدون ترتیب به هم دیگر map می شدند ما از ساختار داده dictionary پایتون استفاده کردیم . که در متد سازنده مقدار خالی گرفته است .

در اینجا نیز ساختار برنامه همانند حالت قبلی است که سزار را امتحان کردیم . یک کلاس با چند متد . همانند حالت قبل عملیات از متد decrypt آغاز می شود . در آنجا دو متد find-key و find-plain-with-key فراخوانی می شوند .

```
def decrypt(self):
        self.find_key()
        self.find_plain_with_key()
        return self.plain
```

نمونه کد ۵: تابع شروع عملیات

تابع find-key :

```
def find_key(self):
        self.find_frequencies()
        self.english_frequency = normalize_dictionary(self.english_frequency)
        self.cipher_frequency = normalize_dictionary(self.cipher_frequency)

        temp = list(self.english_frequency.keys())
        for i, c in enumerate(self.cipher_frequency.keys()):
            self.key[c] = temp[i]
```

نمونه کد ۶: تابع پیدا کردن کلید

همانطور که مشاهده می شود در این تابع چند متد فراخوانی می شود . در متد find-frequencies فرکانس استفاده از حروف مختلف محاسبه می شود . هم اکنون ما نیاز داریم تا بر اساس ترتیب حروف الفبا در دو دیکشنری ، آنها را به هم map کنیم . پس روی همین فرکانس ها و فرکانس حروف در ادبیات انگلیسی متد normalize-dictionary فراخوانی می شود و در نهایت کلید را استخراج کردیم .

در تابع find-frequencies تعداد هر حروف را در متن می شماریم و در یک دیکشنری متعلق به ویژگی های کلاس می ریزیم

```
def find_frequencies(self):
        for c in self.uppercase_list:
            self.cipher_frequency[c] = 0
        for c in self.cipher:
            self.cipher_frequency[c] += 1
```

نمونه کد ۷: پیدا کردن فرکانس ها

هدف متد normalize-dictionary این است که حروف را بر اساس فرکانس هایی که خودمان در اوردیم و فرکانس های واقعی ادبیات انگلیسی به ترتیب map کنیم .

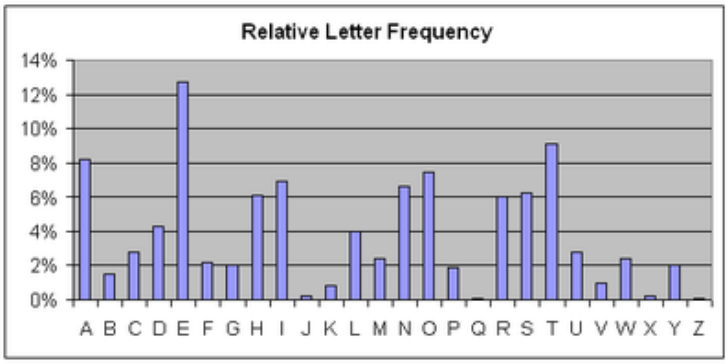
```
def normalize_dictionary(d):
    d = dict(sorted(d.items(), key=lambda item: item[1], reverse=True))

    d_min = min(d.values())
    d_max = max(d.values())
    for k in d:
        d[k] = (d[k] - d_min) / (d_max - d_min)
    return d
```

نمونه کد ۸: متد نرمال کردن فرکانس ها

متد find-plain-with-key در این جا بسیار ساده است زیرا کلید تنها یک دیکشنری است . در نتیجه ما هر کاراکتر در cipher text را به دیکشنری می دهیم و کاراکتر های plain text استخراج می شود . آن ها را کنار هم میگذاریم و باز میگردانیم

```
def find_plain_with_key(self):
        for c in self.cipher:
            self.plain += self.key[c]
        return self.plain
```

نمونه کد ۹: تابع استخراج متن بر اساس کلید



از فضای اینترنت حروف انگلیسی و فرکانس هایشان را استخراج کردیم [۲] سپس از بین حروف cipher text فرکانس کلمات را استخراج کردیم و به ترتیب بیشتر بودن فرکانس یک حرف آن را map کردیم به حروف انگلیسی . [۳]
این روش نیز ما را به جواب نرساند .

۳.۲.۱ مستوی (Affine)

با این روش به جواب رسیدیم . در اینجا برای شکستن رمز از معکوس تابع مستوی استفاده کردیم :

تابع رمزنگاری مستوی :

$$E_b(m) = (a \times m + b) mod 26$$

معکوس تابع رمزنگاری مستوی : به شکل زیر است، m اندیس کلمه رمز شده است . این m همان اندیس کلمه رمز شده است که ضرب و تقسیمی رویش انجام شده و باقی مانده اش بر ۲۶ گرفته شده . در نتیجه ما می آییم و با امتحان کردن مضارب مختلف ۲۶ و اضافه کردنش به m به کلمه رمز شده قبل از باقی مانده گرفتن بر ۲۶ برسیم . (یعنی k را یکی یکی زیاد میکنیم) سپس معکوس ضرب و تقسیم انجام شده را اعمال میکنیم تا به کلمه رمزشده برسیم .

$$D_b(m) = (26 \times k + m - b) \div a$$

۳.۱ ساختار

ما در برنامه یک کلاس داریم به نام DecryptText که سه تابع دارد .

۱.۳.۱ constructor

به هنگام ساخته شدن object ، constructor کلاس صدا زده می شود و مقادیر اولیه را در کلاس مقدار دهی میکنیم . مقادیر اولیه عبارتند از :

- cipher text (از نوع string)
- uppercase list (از نوع list)
- english frequency (از نوع dictionary)

و مقادیر plain text و key خالی هستند چون در طول برنامه محاسبه می شوند .

```
۱ def __init__(self, cipher_text):
۲         self.cipher = cipher_text.replace(' ', '').replace('\n', '')
۳         self.uppercase_list = list(string.ascii_uppercase)
۴         self.key = (0, 0)
۵         self.plain = ''
۶         self.words = []
۷         self.english_words_set = set(words.words())
```

نمونه کد ۱۰: متد سازنده کلاس

۲.۳.۱ توابع

decrypt

بعد از ساخته شدن کلاس ، عملیات کلاس از متد decrypt شروع می شود .

```
def decrypt(self):
        print(started..." "decryption)
        for a in range(1, 26):
            if math.gcd(a, 26) == 1:
                for b in range(1, 26):
                    self.find_plain_with_key(a, b)
                    words = self.find_words()
                    if len(self.cipher) > 2 * len(words):
                        self.words = words
                        self.key = (a, b)
                        break

            if len(self.words) > 0:
                print(,b:"a with "done, self.key)
                break
```

نمونه کد ۱۱: متد آغاز کننده برنامه

در رمزنگاری مستوی دو عدد a و b داریم . برای این ها دو for تو در تو میگذاریم که از ۱ تا ۲۶ را طی میکند . ۲۶ و a نیز باید نسبت به هم اول باشند برای همین در هر iteration ب.م.م a و۲۶ را حساب میکنیم و اگر برابر با یک بود فرآیند iteration را ادامه میدهیم . این کار حالت های تحت بررسی را کم میکند در نتیجه برنامه سریع تر اجرا می شود . در هر iteration تابع find-plain-with-key فراخوانی می شود .

find-plain-with-key

این متد برای هر کلید plain-text را استخراج میکند . در اینجا برای کلید به دست آمده معکوس Affine را میزنیم به کلمه ای که رمز شده است برسیم. بعد از استخراج هر کاراکتر آن را به ویژگی plain text در کلاسمان اضافه میکنیم .

```
def find_plain_with_key(self, a, b):
        self.plain = ''
        for c in self.cipher:
            index = self.uppercase_list.index(c)
            for i in range(30):
                temp = ((index + i*26 - b)/a)
                if temp.is_integer():
                    index = int(temp)
                    break
            self.plain += self.uppercase_list[index]
```

نمونه کد ۱۲: استخراج plain-text برای یک کلید

find-words

به کمک این متد می فهمیم که plain text به دست آمده تا چه حد خوب و معنادار است .

ما قبل تر از مجموعه لغت ها یک set ساختیم . (پایتون عملیات سرچ را روی set بسیار سریع انجام می دهد .)
اما این مسئله این است که کدام بخش از متن را سرچ کنیم . برای اینکار از یک الگوریتم dynamic programming استفاده میکنیم . این الگوریتم به این صورت کار میکند که ابتدا یک حرف را در نظر میگیرد و بررسی میکند که آیا کلمه معنی دار هست یا نه . این الگوریتم یک حرف تنها را کلمه معنا دار میگیرد . سپس تا یک عمق بیشینه ای (در اینجا ۱۵) حرف را یکی یکی جلو می برد و در هر مرحله بررسی میکند که آیا کلمه به دست آمده کلمه با معنی هست یا نه . عمق بیشینیه برای این ۱۵ است که ما بیشینه طول کلمات را ۱۵ فرض کردیم .
سپس برای هر plain-text که به دست می آید تعداد کلمات را حساب میکنیم .

در حالتی که طول همه کلمه ها دو باشد (ما تقریبا کلمه ای با طول یک حرف نداریم در نتیجه طول کمترین حالت ممکن است.) باید به تعداد نصف کاراکتر های متن کلمه داشته باشیم . یعنی حداکثر تعداد کلمات ، نصف تعداد کاراکتر های cipher-text است . در نتیجه یک if گذاشتیم و شرط بالا را روی تعداد کلمات در نظر گرفتیم .
تنها در یک حالت بود که این شرط برآورده می شد .
در حالتی که این شرط برآورده می شد طول وِیژگی words کلاس که از جنس لیست است از ۰ بیشتر می شد و همین به عنوان شرط توقف الگوریتم ما در نظر گرفته شد .

```
۱  def find_words(self, max_depth=15):
۲          word_first_index = 0
۳          words = []
۴          while word_first_index < len(self.plain):
۵              word_last_index = word_first_index
۶              for j in range(max_depth):
۷
۸                  word_last_index = word_first_index + j + 1
۹                  searched_word = self.plain[word_first_index:word_last_index]
۱۰                 searched_word = searched_word.lower()
۱۱
۱۲                 if searched_word in self.english_words_set:
۱۳                     word_last_index = word_first_index + j + 1
۱۴
۱۵             words.append(self.plain[word_first_index:word_last_index])
۱۶             word_first_index = word_last_index
۱۷         return words
```

نمونه کد ۱۳: متد آغاز کننده برنامه

۴.۱ خروجی نهایی

متن ورودی

متن رمز شده به شکل زیر بود

CIMWB DWQPW TPAMS DDAJP TKPKJ KPWMA ZPWJK NEPWD WJMWQ
AXTPF IMRWV WRAZQ KDKQI PEKXT RKPWX QEYIR RNWSX DJWQW
TWXPW TKXTY IRRWB PWXTP FWDWJ ZAJUK XQWAZ CKDDR IQKPI
AXMKR AXCYI PFWBD KXTIX CPFWM QADWA ZQKDK NIRIP IWMIX
MSDDA JPAZI XQJWK MIXCR EXWYK XTIXX AVKPI VWKDD RIQKP
IAXMK QJAMM PFWJW KRUMA ZYIJW RWMMQ AXXWQ PIVIP EQACX
IPIAX MWXMI XCKXT IUKCI XCCFI CFWJZ JWGSW XQIWM YIRRW
XKNRW USQFZ KMPWJ MKUDR IXCJK PWMIX KTTIP IAXPA DJAVI
TIXCM ICXIZ IQKXP RENWP PWJPF JASCF DSPKX TFICF WJTKP
KJKPW MPFWQ AUNIX KPIAX AZMSN UUYKV WWCYK VWRWX CPFMM
UKRRW JPFKX AXWUI RRIUW PWJKX TPFWS MWAZZ JWGSW XQEMW
RWQPI VIPEP ATWPW JUIXW JWRKP IVWWR WQPJA UKCXW PIQKN
MAJDP IAXJK PWMIM WBDWQ PWTPA RWKTP ADAPW XPIKR REMIC
XIZIQ KXPKT VKXQW MIXYI JWRWM MMWXM IXCPW QFXAR ACEKT
TIPIA XKRRE YFWJW KMPFW KTTIP IAXAZ UANIR WWTCW QAUDS
PIXCI MKDAI XPAZQ AXMIT WJKPI AXKMK XKTTI PIAXP ACXWP
YAJOM UANIR WWTCW QAUDS PIXCY IRRNW NSIRP IXPAK RRCXW
PYAJO MWTCW KXTQA JWQAU DSPIX CYIRR NWQAU WUSQF UA-
JWM WKURW MMREI XPWCJ KPWTK MDKJP AZKQA UNIXW TQAUU
SXIQK PIAXM QAUDS PKPIA XIXZJ KMPJS QPSJW ZJKUW YAJON
EPFWP IUWCX WPYAJ OMKJW TWDRA EWTPF IMYIR RDJAV ITWUK
XEDAP WXPIK RKTVK XPKCW MKMCP WQFXA RACEN WQAUW
MADWJ KPIAX KRIXQ RSTIX CIUDJ AVWTK QQWMM PAKJP IZIQI
KRIXP WRRIC WXQWQ KDKNI RIPIWM

متن خروجی

بعد از شکستن رمز به متن زیر رسیدیم :

G IS EXPECT E D TO SUPPORT DATA RATE SOFT ERA BY TE PERSE COND THIS LEVEL OF CAPACITY AND LATENCY WILL BE UNPRECEDENTED AND WILL EXTEND THE PERFORMANCE OF GAP PLICATION SALON G WITHE X PAND ING THE SCOPE OF CAP A B I LIT IE SIN SUPPORT OF INCREASINGLY NEW AND INNOVATIVE APPLICATION SACRO S ST HERE ALMS OF WIRELESS CONNECTIVITY COGNITION SEN SING AND I MAGI N G G HIGHER F RE Q U EN C IE SWILL ENABLE MUCH FASTER SAMPLING RATE SINA D DIT ION TOP R O V ID ING SIGNIFICANTLY BETTER THROUGHPUT AND HIGHER DATA RATE ST HE COMBINATION OF SUB M M WAVE E G WAVE LENGTH S SMALLER THAN ONE MILLIMETER AND THE USE OFF RE Q U EN C Y SELECTIVITY TODE TERMINER ELATIVE ELECTROMAGNETIC ABSORPTION RATE SISE X P E C TED TOLE AD TOPO TENT I ALLY SIGNIFICANT ADVANCE SIN WIRELESS SEN SING TECHNOLOGY ADDITIONALLY WHEREAS THE ADDITION OF MOBILE EDGE C OM PUT ING IS A POINT OF CONSIDERATION ASANA D DIT ION TOG NETWORK S MOBILE EDGE C OM PUT ING WILL BE BUILT INTO ALL G NETWORK SEDGE AND CORE C OM PUT ING WILL BECOME MUCH MORES EA M LESS LY INTEGRATE DAS PARTO FA COMBINED COMMUNICATION S COMPUTATION INFRASTRUCTURE FRAMEWORK BYTH E TIME G NETWORK SARE DEPLOY E D THIS WILL PROVIDE MANY POTENTIAL ADVANTAGE SA S G TECHNOLOGY BECOMES OPERATIONAL IN C L U DING IMPROVE DA C CESS TOA R TI FI C I ALIN TELL I GEN CE CAPABILITIES

در پایان با وجود اینکه به plain text رسیدیم، به دلیل اینکه بعضی از کلمات در پکیج nltk.corpus وجود ندارند این کد بهترین خروجی را نمیدهد. به طور مثال کلمه expect در آن موجود است ولی expected نیست. به همین دلیل خروجی کد را به صورت دستی تصحیح می کنیم. این متن در رابطه با نسل ششم شبکه

تلفن همراه است.

G IS EXPECTED TO SUPPORT DATA RATES OF TERABYTE PER SECOND. THIS LEVEL OF CAPACITY AND LATENCY WILL BE UNPRECEDENTED AND WILL EXTEND THE PERFORMANCE OF G APPLICATIONS ALONG WITH EXPANDING THE SCOPE OF CAPABILITIES IN SUPPORT OF INCREASINGLY NEW AND INNOVATIVE APPLICATIONS ACROSS THE REAL MS OF WIRELESS CONNECTIVITY COGNITION SENSING AND IMAGING. G HIGHER FREQUENCIES WILL ENABLE MUCH FASTER SAMPLING RATES IN ADDITION TO PROVIDING SIGNIFICANTLY BETTER THROUGHPUT AND HIGHER DATA RATES. THE COMBINATION OF SUB MM WAVE E.G WAVE LENGTHS SMALLER THAN ONE MILLIMETER AND THE USE OF FREQUENCY SELECTIVITY TO DETERMINE RELATIVE ELECTROMAGNETIC ABSORPTION RATES IS EXPECTED TO LEAD TO POTENTIALLY SIGNIFICANT ADVANCES IN WIRELESS SENSING TECHNOLOGY ADDITIONALLY WHEREAS THE ADDITION OF MOBILE EDGE COMPUTING IS A POINT OF CONSIDERATION. AS AN ADDITION TO G NETWORKS MOBILE EDGE COMPUTING WILL BE BUILT INTO ALL G NETWORKS EDGE AND CORE COMPUTING WILL BECOME MUCH MORE SEAMLESSLY INTEGRATED AS PART OF A COMBINED COMMUNICATIONS COMPUTATION INFRASTRUCTURE FRAMEWORK BY THE TIME G NETWORKS ARE DEPLOYED. THIS WILL PROVIDE MANY POTENTIAL ADVANTAGES AS G TECHNOLOGY BECOMES OPERATIONAL INCLUDING IMPROVED ACCESS TO ARTIFICIAL INTELLIGENCE CAPABILITIES.

Bibliography

[1] Barthelemy, "How to check if a word is an english word with python?." https://stackoverflow.com/questions/3788870/how-to-check-if-a-word-is-an-english-word-with-python.

[2] unknown, "The frequency of the letters of the alphabet in english." https://www3.nd.edu/~busiforc/handouts/cryptography/letterfrequencies.html.

[3] R. Jedynak, "How do i decode monoalphabetic cipher with unknown keyword?." https://www.researchgate.net/post/How-do-I-decode-monoalphabetic-cipher-with-unknown-keyword.