دانشکده مهندسی کامپیوتر

امنیت سیستم‌های کامپیوتری

زمستان ۱۴۰۰

پروژه اول

توسعه سامانه شکستن رمزهای کلاسیک

آخرین ویرایش ۲۹ اسفند ۱۴۰۰

اعضای گروه:

علی صداقی ارقون ۹۷۵۲۱۳۷۸

کیوان داداش‌زاده ۹۷۵۲۲۱۴۸

محمد یارمقدم ۹۶۴۶۲۱۰۴

فهرست مطالب

۱ مقدمه ۲

۲ روش‌های موجود ۲

۱.۲ کورکورانه ۲

۲.۲ سزار ۲

۳.۲ مستوی ۳

۴.۲ ویگنر ۳

۵.۲ تحلیل فرکانسی ۴

۶.۲ الگوریتم‌های ژنتیک ۴

۳ توضیحات کد ۵

۱.۳ بارگذاری فایل فرکانس کلمات و محاسبه هزینه فاصله گذاری ۵

۲.۳ بارگذاری متن رمز شده ۵

۳.۳ محاسبه ب.م.م ۶

۴.۳ هسته اصلی برنامه و رمزگشای مستوی ۶

۵.۳ تابع درایور برنامه و آرگومان‌ها ۷

۶.۳ تابع کمکی برای اسپیس‌گذاری ۸

۴ نحوه اجرای برنامه ۹

۵ ورودی برنامه ۱۰

۶ نتایج ۱۲

۷ منابع ۱۴

۱ مقدمه

در این پروژه قصد داریم با ورودی گرفتن یک متن رمز شده یا Cipher Text آن را رمزگشایی کنیم و به متن اصلی یا همان Plain Text برسیم.

این نوع رمزگشایی ما در دسته حمله بر اساس فقط متن رمز شده یا به عبارت دیگر Ciphertext Only Attack قرار می‌گیرد که از آن تحت عنوان حمله نوع اول نیز یاد می‌شود.

۲ روش‌های موجود

به صورت کلی برای رمزگشایی که متن رمز شده بایستی روش‌های رمزنگاری را بدانیم و با مهندسی معکوس عکس عمل آن‌ها را انجام دهیم. سپس نتیجه به دست آمده را بررسی کنیم و مشخص کنیم که آیا به متن اصلی دست یافته ایم یا خیر. در واقع با یک فرایند تکراری و چرخشی در طرف هستیم که با آزمون و خطا بسیار همراه است.

با توجه به اینکه این پروژه برای شکستن یک رمز کلاسیک است تنها روش‌های تدریس شده در کلاس درس را بررسی می‌کنیم.

۱.۲ کورکورانه

منظور از روش کورکورانه همان Brute Force است. در این روش تمامی فضای حالت مسئله مورد بررسی قرار می‌گیرد و تمامی حالت‌های ممکن آزموده می‌شود.

این روش به دلیل بزرگی فضای حالت بسیار زمان‌بر است و عملا در این پروژه کاربردی نداشت. در تمرین اول درس تخمینی از زمان مورد نیاز در این روش انجام دادیم و همانطور که مشاهده شد زمان رمزگشایی حتی می‌تواند تا قرن‌ها طول بکشد.

استفاده از این روش هنگامی که هیچ اطلاع قبلی از نوع رمزگذاری نداریم می‌تواند مفید باشد.

۲.۲ سزار

الگوریتم سزار در واقع یک جانشینی ساده تک حرفی است و ترتیب جانشینی همواره ثابت است، یعنی میان هر حرف با حرف جایگزین یک مقدار ثابت فاصله وجود دارد. حالت کلی‌تر این روش در قسمت

پایین آورده شده است. ما هم بدلیل وجود روش پایین از روش سزار استفاده نکردیم.

رابطه کلید در رمزنگاری سزار به صورت زیر است:

$$c = E_k(m) = (m + k) \mod 26$$

۳.۲ مستوی

رمز سزار بسیار ساده بود و به سادگی قابل شکسته شدن بود. بنابراین روش کلی‌تر آن یعنی رمز مستوی (Affine Cipher) ارائه شد.

رابطه کلید در آن به صورت زیر است:

$$c = E_k(m) = (a \times m + k) \mod 26$$

تابع رمزگشایی:

$$p = D_k(m) = \frac{(26 \times m - b)}{a}$$

در این روش با دو متغیر a و k در طرف هستیم پس قدرت الگوریتم بیشتر از سزار است. نوع کلی‌تر این الگوریتم نیز موجود است که طول کلید در آن ۸۸ بیت است!

همانطور که در کلاس درس استاد مطرح کردند، این متن توسط این الگوریتم رمز شده است و تنها پیاده‌سازی این روش برای رمزگشایی کافی است.

۴.۲ ویگنر

رمزنگاری ویگنر یا Vignere Cipher نوعی رمزنگاری چند الفبایی است. رمزنگاری در این روش با استفاده از یک جدول صورت می‌گیرد. حروف متن اصلی تعیین کننده ستون و حروف عبارت کلید تعیین کننده سطر است.

برای رمزنگاری این مسئله از این روش استفاده نشده است و اگر با مهندسی معکوس عکس آن را انجام دهیم تا رمزگشایی صورت بگیرد به نتیجه مطلوبی نخواهیم رسید زیرا رمزنگاری ما چند الفبایی نیست.

۵.۲ تحلیل فرکانسی

این روش برای شکستن رمزنگاری‌های جانشینی بسیار مناسب است. مبنای کار آن تکرار حروف در الفبای انگلیسی است. ابتدا تکرار هر حرف الفبا در متن رمز شده را به دست می‌آوریم. سپس تکرار حروف الفبای انگلیسی در متن‌های بزرگ را به دست می‌آوریم و بر اساس تعداد تکرار مرتب می‌کنیم. سپس میان این دو لیست یک نگاشت برقرار می‌کنیم.

این روش تا حد خوبی می‌تواند مسئله ما را رمزگشایی کند . اما به جواب مطلق درست نخواهد رسید. قسمتی از این روش درون فایل freq-analysis.py پیاده‌سازی شده است. اما به دلیل صحبت استاد در کلاس درس که روش Affine کافی است دیگر پیاده‌سازی این بخش را تکمیل نکردیم.

۶.۲ الگوریتم‌های ژنتیک

یکی از روش‌های معمول برای رمزگشایی استفاده از الگوریتم‌های ژنتیک و تکاملی است. در این روش‌ها که مبتنی بر آزمون و خطا است، با ایجاد نسل‌های (Generations) مختلف و محاسبه Fitness آن‌ها در محیط سعی می‌کنیم در جهت بهترین نسل حرکت کنیم.

نسل‌های معیوب توسط محیط حذف می‌شوند و نسل‌های مناب با هم ترکیب می‌شوند یا دچار جهش (Mutation) می‌شوند تا نسل بهتری ایجاد کنند.

این روش بدلیل پیچیدگی زیادی که داشت و همچنین قطعی نبودن نتیجه آن مورد بررسی عملی قرار نگرفت.

۳ توضیحات کد

در این قسمت تصاویری از توابع موجود در برنامه خواهد آمد و در ادامه‌ی توضیح مختصری درباره عملکرد آن تابع داده خواهد شد.

۱.۳ بارگذاری فایل فرکانس کلمات و محاسبه هزینه فاصله گذاری

```
9  def load_words_from_file_and_put_score(freq_words_path: Path):
10
11     if not freq_words_path.exists():
12         raise FileFileNotFoundError
13
14     with freq_words_path.open('r') as words_file:
15         words = words_file.read().split() # extract words from file path
16         wordcost = dict((k, log((i+1)*log(len(words)))) for i,k in enumerate(words))
17         maxword = max(len(x) for x in words)
18
19     return wordcost, maxword
```

شکل ۱: کد فرکانس کلمات و محاسبه هزینه

این تابع فایل تکرار کلمات را بارگذاری می‌کند و بر اساس ترتیب کلمات یک امتیاز یا هزینه نسبت به هر کلمه اختصاص می‌دهد. طبیعتا هر چه تکرار یک کلمه بیشتر باشد برای ما مطلوب‌تر است که آن کلمه را در متن داشته باشیم و اسپیس گذاری را بر مبنای آن انجام می‌دهیم.

۲.۳ بارگذاری متن رمز شده

```
21 def load_cipher_text(cipher_text_path: Path):
22
23     if not cipher_text_path.exists():
24         raise FileFileNotFoundError
25
26     with cipher_text_path.open('r') as cipher_file:
27         cipher_text = cipher_file.read()
28         return cipher_text
```

شکل ۲: بارگذاری متن رمز شده

این تابع پیچیدگی زیادی ندارد و تنها فایل متن رمز شده را بارگذاری می‌کند.

۳.۳ محاسبه ب.م.م

```
30  def gcd(a, b):
31      while a != 0:
32          a, b = b % a, a
33      return b
```

شکل ۳: محاسبه ب.م.م

این قسمت برای بررسی قابل تقسیم پذیر بودن دو عدد است. به فرمول رمزگشایی بخش مستوی توجه شود.

۴.۳ هسته اصلی برنامه و رمزگشای مستوی

```
35  def break_affine(cipher_text: str, split_space_func):
36      print(f"[*] Start breaking cipher text:\n{cipher_text}")
37      plain_texts = []
38      for z in range(26):
39          if gcd(z, 26) != 1:
40              continue
41          for x in range(26):
42              prob_cipher = []
43              word_dicts = {}
44              prob_cipher = [alphabet[(i*z+x)%26] for i in range(26)]
45              for cnt in range(26):
46                  word_dicts[prob_cipher[cnt]] = alphabet[cnt]
47              plain_texts.append(split_space_func(cipher_text.translate(str.maketrans(word_dicts)).lower()))
48      max_of_plain_text = len(cipher_text)*2
49      main_plain_text_index = 0
50      for cnt in range(len(plain_texts)):
51          if max_of_plain_text > len(plain_texts[cnt]):
52              max_of_plain_text = len(plain_texts[cnt])
53              main_plain_text_index = cnt
54      print(f"[*] Decrypted Plain text: \n{plain_texts[main_plain_text_index]}")
```

شکل ۴: رمزگشای مستوی

در این قسمت طبق فرمولی که برای رمزگشایی مستوی ارائه دادیم عمل می‌شود. هر بار که کلید

جدیدی ساخته می‌شود با انجام فاصله گذاری‌های مختلف سعی می‌شود بهترین فاصله گذاری به ازای آن کلید پیدا شود. سپس میزان بهینه بودن آن متن پیدا شده محاسبه شده و نگه داشته می‌شود. هنگامی که متنی با بهینگی بهتر پیدا شد آن را به عنوان متن بهتر ذخیره می‌کنیم. در نهایت بهترین متن خروجی داده می‌شود.

۵.۳ تابع درایور برنامه و آرگومان‌ها

```
57  if __name__ == "__main__":
58      parser = argparse.ArgumentParser()
59      parser.add_argument('-c', '--cipher-path', help="Path to cipher text file", type=str, required=True, dest="cipher_text_path")
60      parser.add_argument('-f', '--freq-words-path', help="Path to frequency words", type=str, required=True, dest="freq_words_path")
61      parser.add_argument('-s', '--speed', help="Speed up breaking process(only for brute force)", action='store_true', required=False, dest="speed_up")
62      args = parser.parse_args()
63
64      if args.speed_up:
65          from proj_utils import utils
66          wordcost, maxword = load_words_from_file_and_put_score(Path(args.freq_words_path))
67          split_space_func = lambda string: utils.infer_spaces(string, wordcost, maxword)
68      else:
69          try:
70              import wordninja
71              split_space_func = lambda string: " ".join(wordninja.split(string))
72          except ImportError:
73              print("[*] Cannot import wordninja. install wordninja package or use speed up option")
74              exit()
75
76      cipher_text = load_cipher_text(Path(args.cipher_text_path))
77      break_affine(cipher_text, split_space_func)
```

شکل ۵: درایور برنامه و آرگومان‌ها

در این قسمت آرگومان‌های برنامه تعریف می‌شوند تا بر اساس آن‌ها برنامه راهنمای اجرا نیز داشته باشد. بر اساس پرچم S- مشخص می‌شود اسپیس گذاری به چه نحوی صورت بگیرد. دو روش برای فاصله گذاری ایجاد شده.

۱. استفاده از فایل فرکانس کلمات که روش سریع‌تری با دقت کمتر است.

۲. استفاده از کتابخانه wordninja که روشی دقیق‌تر اما آهسته‌تری است. برای این روش شما باید این کتابخانه را نصب داشته باشید.

۶.۳ تابع کمکی برای اسپیس‌گذاری

```python

# copied this function from
# https://exchangetuts.com/how-to-split-text-without-spaces-into-list-of-words-1639498085014014

from math import log

def infer_spaces(s: str, wordcost: dict, maxword: int):
    """Uses dynamic programming to infer the location of spaces in a string
    without spaces."""

    # Find the best match for the i first characters, assuming cost has
    # been built for the i-1 first characters.
    # Returns a pair (match_cost, match_length).
    def best_match(i):
        candidates = enumerate(reversed(cost[max(0, i-maxword):i]))
        return min((c + wordcost.get(s[i-k-1:i], 9e999), k+1) for k,c in candidates)

    # Build the cost array.
    cost = [0]
    for i in range(1,len(s)+1):
        c,k = best_match(i)
        cost.append(c)

    # Backtrack to recover the minimal-cost string.
    out = []
    i = len(s)
    while i>0:
        c,k = best_match(i)
        assert c == cost[i]
        out.append(s[i-k:i])
        i -= k

    return " ".join(reversed(out))
```

شکل ۶: تابع کمکی برای اسپیس‌گذاری

این قسمت از یک اینترنت برداشته شده است و برای ایجاد فاصله در یک رشته بدون فاصله بکار می‌رود. لینک آن در قسمت منابع موجود است.

نحوه کارکرد آن به صورت برنامه‌ریزی پویا (Dynamic Programming) است. به صورتی که طول‌های مختلفی برای جداسازی کلمات استفاده می‌شود و در هر مرحله یک هزینه (Cost) محاسبه می‌شود. در نهایت بهترین نتیجه خروجی داده می‌شود.

۴ نحوه اجرای برنامه

برای راحتی اجرای برنامه یک Help ایجاد شده است که با زدن دستور زیر می‌توانیم آن را مشاهده کنیم.

python main.py -h

```
C:\Users\alise\Documents\University\Semesters\Term 8\_Security\Homeworks\CS_P1\project1>python main.py -h
usage: main.py [-h] -c CIPHER_TEXT_PATH -f FREQ_WORDS_PATH [-s]

optional arguments:
  -h, --help            show this help message and exit
  -c CIPHER_TEXT_PATH, --cipher-path CIPHER_TEXT_PATH
                        Path to cipher text file
  -f FREQ_WORDS_PATH, --freq-words-path FREQ_WORDS_PATH
                        Path to frequency words
  -s, --speed           Speed up breaking process(only for brute force)
```

شکل ۷: راهنمای اجرای برنامه

نمونه دستور برای اجرا برنامه:

CLI Commands:

python main.py -c cipher_text.txt -f words-by-frequency.txt -s

python main.py -c cipher_text.txt -f words-by-frequency.txt

دستور اول برای اسپیس‌گذاری از فایل کلمات پر تکرار استفاده می‌کند که روش سریع‌تری است. دستور دوم از کتابخانه wordninja برای فاصله‌گذاری صحیح میان کلمات استفاده می‌کند که نیازمند نصب بودن این کتابخانه است. این روش سرعت کمتری دارد.

۵ ورودی برنامه

با حذف اسپیس‌های دکوری از متن رمز شده (Cipher Text) درون اسلاید درس و ذخیره آن در فایل cipher_text.txt آن را آماده اجرا در برنامه می‌کنیم.

این متن به صورت زیر است:

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 KT-TIP 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 MA-JDP IAXJK PWMIM WBDWQ PWTPA RWKTP ADAPW XPIKR REMIC XIZIQ KXPKT VKXQW MIXYI JWRWM MMWXM IX-CPW 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 UAJWM WKURW MMREI XPWCJ KPWTK MDKJP AZKQA UNIXW TQAUU SX-

IQK PIAXM QAUDS PKPIA XIXZJ KMPJS QPSJW ZJKUW YA-
JON 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

۶ نتایج

با اجرای برنامه و به دست آمدن مقدار صحیح کلید یعنی مقادیر a و k و اسپیس گذاری اتوماتیک صحیح به متن اصلی می‌رسیم.

```
C:\Users\alise\Documents\University\Semesters\Term 8\_Security\Homeworks\CS_P1\project1>python main.py -c cipher_text.txt -f words-by-frequency.txt -s
[*] Start breaking cipher text:
CIMWBDWQPWTPAMSDDAJPTKPKJKPWMAZPWJKNEPWDWJMWQAXTPFIMRWVRAQKKQPEXTKPXQYIRNSXJWWTXPTKTYRRBPWXTPFWDWJZAJUKXQWAZCKDDRIQKPIAXMKRAXCYIPFWBDKXTIXCPFWMQADWAZQKDKNIR
IPIWMIXMSDDAJPAZIXQJWKMIXCREXWYKXTIXXAVKPIVWKDDRIQKPIAXMKQJAMMPFWJWKRUMAZYIJWRWMMQAXXWQPIVIPEQACXIPIAXMWXMIXCKXTIUKCIXCCFICFWJZJWGSWXQIWMYIRRWXKNRWUSQFZKMPW
JMKUDRIXCJKPWMIXKTTIPIAXPADJAVTIXCMICXIZIQKXPRENWPPWJPFJASCFDSPKXTFICFWJTKPKJKPWMPFWQAUNIKPIAXAZMSNUUYKVWWCYKVWRWXCPFMMUKRRWJPFKXAXWUIRRIUWPWJKXTPFWSMWAZZJW
GSWXQEMWRWQPIVIPEPATWPWJUIXWJWRKPIVWWRWQPJAUKCXWPIQKNMAJDPIAXJKPWMIMWBDWQPWTPARWKTPADAPWXPIKRREMICXIZIQKXPKTVKXQWMIXYIJWRWMMMWXMIXCPWQFXARACEKTTIPIAXKRREYFW
JWKMPFWKTTIPIAXAZUANIRWWTCWQAUDSPIXCIMKDAIXPAZQAXMITWJKPIAXKMKXKTTIPIAXPACXWPYAJOMUANIRWWTCWQAUDSPIXCYIRRNWNSIRPIXPAKRRCXWPYAJOMWTCWKXTQAJWQAUDSPIXCYIRRNWQA
UWUSQFUAJWMWKURWMMREIXPWCJKPWTKMDKJPAZKQAUNIXWTQAUUSXIQKPIAXMQAUDSPKPIAXIXZJKMPJSQPSJWZJKUWYAJONEPFWPIUWCXWPYAJOMKJWTWDRAEWTPFIMYIRRDJAVITWUKXEDAPWXPIKRKTVK
XPKCWMKMCPWQFXARACENWQAUWMADWJKPIAXKRIXQRSTIXCIUDJAVWTKQQWMMPAKJPIZIQIKRIXPWRRICWXQWQKDKNIRIPIWM

[*] Decrypted Plain text:
g is expected to support data rates of terabyte per second this lev loc a act y nd at nc wil bun reed nt dad w ll x tend the performance of g applications a
long with expanding the scope of capabilities in support of increasingly new and innovative applications across the realms of wireless connectivity cognitio
n sensing and imaging g higher frequencies will enable much faster sampling rates in addition to pro v ding significantly better throughput and higher data
rates the combi ation of sub mm wave eg wavelengths smaller than one millimeter and the use of frequency selectivity to determine relative electromagnetic a
bsorption rates is expected to lead to potentially significant advances in wireless sensing technology additionally whereas the addition of mobile edge comp
uting is a point of consideration as an addition to g networks mobile edge computing will be built into allg networks edge and core computing will become mu
ch more seamless ly integrated as part of a combined communications computation infrastructure framework by the time g networks are deployed this will provi
de many potential advantages as g technology becomes operational including improved access to artificial intelligence capabilities
```

شکل ۸: خروجی برنامه

این متن به صورت زیر 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 realms 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 eg wavelengths 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

نکته: برخی از خطاهایی که در تصویر آورده شده است بدلیل فاصله گذاری اشتباه الگوریتم بوده است. که به سادگی می‌توان آن‌ها را به صورت دستی اصلاح کرد. با توجه به توضیحات استاد نیازی به اسپیس گذاری صحیح نبود اما این بحش نیز توسط ما پیاده‌سازی شد. همچنین استفاده از روش wordninja برای اسپیس‌گذاری نتیجه بهتری خواهد داشت.

۷ منابع

لینک پروژه لاتک درون فایل LaTeX_Link.txt موجود است.

How to split text without spaces into list of words

Word Ninja Library

Genetic algorithm for cryptography

English words frequencies