CMSC 626 Principles of Computer Security

Assignment 1

Faisal Rasheed Khan

VB02734

Vb02734@umbc.edu

1.

Cipher Text:

V@ocllzfc@ph@z@yitc@ph@z@bpjuocsw@ph@lpoc@pwgch@jpttcjwips@py@bzwz@il@lzib@w
p@ec@zuwgcswij@agcs@iw@il@fcsuisc@zsb@jzoc@yhpo@iwl@zttcfcb@lpuhjc_@Jcllzfc@ph@bz
wz@zuwgcswijzwips@il@z@xhpjcbuhc@wgzw@zttpal@jpoousijzwisf@xzhwicl@wp@kchiyd@wgz
w@hcjcikcb@ph@lwphcb@ocllzfcl@zhc@zuwgcswij@i_c_@jzoc@yhpo@phifiszt@lpuhjc_@Ugc@
wap@ioxphwzsw@zlxcjwl@zhc@wp@kchiyd@wgzw@wgc@jpswcswl@py@wgc@ocllzfc@gzkc@sp
w@eccs@ztwchcb@zsb@wgzw@wgc@lpuhjc@il@zuwgcswij_@Fc@ozd@ztlp@ailg@wp@kchiyd@
wioctiscll@py@z@ocllzfc@zsb@lcnucsjc@hctzwikc@wp@pwgch@ocllzfcl@ytpaisf@ecwaccs@wap
@xzhwicl_@Vtt@py@wgclc@jpsjchsl@jpoc@usbch@wgc@jzwcfphd@py@bzwz@iswcfhiwd_

Key mapping (ciphertext->plaintext) for max occurring character
c -> e

Plain Text:
A message or a file or a document or some other collection of data is said to be authentic when it is genuine and came from its alleged source. Message or data authentication is a procedure that allows communicating parties to verify that received or stored messages are authentic i.e. came from original source. The two important aspects are to verify that the contents of the message have not been altered and that the source is authentic. We may also wish to verify timeliness of a message and sequence relative to other messages flowing between two parties. All of these concerns come under the category of data integrity.

2.
Substitution Keys:

a -> w
b -> d
c -> e
d -> y
e -> b
f -> g
g -> h
h -> r
i -> i
j -> c
k -> v
l -> s
n -> q
o -> m
p -> o

s -> n
t -> l
u -> u
w -> t
x -> p
y -> f
z -> a
@ ->
_ -> .
F -> W
J -> M
U -> T
V -> A

3.
In this substitution cipher, SPACE is substituted by ‘@’ and DOT(‘.’) is substituted by ‘_’. The letter ‘c’ is substituted by ‘e’ and so on. Doing a frequency analysis in ciphertext, three most common character occurrences are

‘c’: 77 times

‘w’: 55 times

‘z’: 47 times

‘p’: 42 times

‘l’: 38 times

‘i’: 34 times

Thus, these letters are likely to be substituted from ‘e’, ‘a’, ‘i’, ‘o’, ‘u’. Based on these trial and error, replace other letters similarly.

‘c’: 77 times	replaced by e
‘w’: 55 times	replaced by z
‘z’: 47 times	replaced by a
‘p’: 42 times	replaced by o
‘l’: 38 times	replaced by s
‘i’: 34 times	replaced by i

And the remaining letters are replaced based on trial and error.

4.

The challenges faced here were to identify the two-letter word as in this cipher text they were more and with this issue, trial and error for the entire cipher text was difficult.

There are more 2 letter words in this cipher text like or, of, to, be, in, is. The first step I replaced single letter with article ‘a’ and tried all the possible 2 letter words replacing. And then with the replacements of 2 letter words I have tried trial and error on 3 letter words and so on.

CMSC 626 Principles of Computer Security

Assignment 2

Faisal Rasheed Khan

VB02734

vb02734@umbc.edu

1.

```
cp /etc/john/john.conf .john/john.conf
cp shadow /home/Faisal
cat shadow
john --wordlist= CMSC626-Enroll-2023-02-10.csv shadow
john --wordlist= CMSC626-Enroll-2023-02-10.csv --rule shadow
john --wordlist= permGokanakonda.txt shadow
john --wordlist= permGokanakonda.txt --stdout

john shadow –show

touch permGokanakonda.txt
touch perm.py
nano perm.py
```

Python Code to generate all permutations:

```
import itertools
alph = "Gokanakonda"
permutations = itertools.product(*[(c.upper(), c.lower()) for c in alph])

with open("/home/Faisal/permGokanakonda.txt", "w") as f:
    for p in permutations:
        f.write("".join(p) + "\n")
```

Python Code to generate permutations for the end numbers:

```
import itertools

name = 'Nandikonda'

numbers = '123456789'

n_count = 2                    //append n number of digits at the end of the username


result = []

for n in itertools.product(numbers, repeat=n_count):

   name_with_numbers = name + ''.join(n)

   result.append(name_with_numbers)


with open('/home/Faisal/permNandikonda2.txt', 'w') as file:

   for name in result:

      file.write(name + '\n')
```

4.

- The challenges faced here are as we don’t have access to use sudo, we need to copy john.conf to our current working directory.
  `cp /etc/john/john.conf .john/john.conf`
- we don’t have access to assign value to the rule in:
  `john --wordlist= CMSC626-Enroll-2023-02-10.csv --rule shadow`
  so we can copy john.conf file and use the command
  `nano .john/john.conf`
  and make custom rule in the wordlist section.
- Some of the username criteria match exceeds the memory and will take a very long time to run. Tried matching those usernames by partial/Full matching using python code by generating passwords according to the mentioned criteria.

References:

CH02-CompSec4e_accessible_L03 (blackboardcdn.com)

John the Ripper - wordlist rules syntax (openwall.com)

Comprehensive Guide to John the Ripper. Part 5: Rule-based attack - Ethical hacking and penetration testing (miloserdov.org)

Python - Itertools.Product() - GeeksforGeeks

itertools — Functions creating iterators for efficient looping — Python 3.11.2 documentation