Lab 1 – Classical Cryptography

Lab introduction

This lab focuses on implementation of simple classical cryptography, including Caesar, Monoalphabetic ciphers and two simple cryptanalytic attacks, including the Brute-Force and Frequency Analysis. After this lab, students are expected to be familiar with the basic concept of encryption, decryption and cipher attacks.

The lab has three problems, and you should follow the order (from 1 to 3) to solve all problems.

This lab exercise shall be done individually.

This lab will be done in 4 sessions, including 1 session for lecture + 1 session is for report preparation.

To perform the lab, basic C programming skill is required.

Problem 1 (50 points)

In this problem, you are required to implement (C) functions for encryption/decryption with the Caesar cipher, as well as attacks.

a) **(20 points)** Implement encryption/decryption functions that take a key (as an integer in 0, 1, 2, … , 25), and a string. The function should only operate on the characters ‘a’, ‘b’, … ‘z’ (both upper and lower case), and it should leave any other characters, unchanged.
   - Provided code: Prob1a_skeleton.c
      + char* CaesarEncrypt(int key, char *plaintext): encryption function that takes a key and a string (plaintext) and returns the ciphertext (string)
      + char* CaesarDecrypt(int key, char *ciphertext): decryption function that takes a key and a string (ciphertext) and returns the plaintext (string)

   Note: you can implement additional functions.

b) **(15 points)** Implement a function that performs a **brute force attack** on a ciphertext, it should print a list of the keys and associated decryptions. It should also take an optional parameter that takes a substring (keyword) and only prints out potential plaintexts that contain that keyword.
   - Provide code: Prob1b_skeleton.c
      + char* CaesarDecrypt(int key, char *ciphertext): same to Problem1.a
      + void BruteForceAttack(char *ciphertext, char *keyword): brute force attack function that takes a ciphertext (string) and a keyword (string). If the keyword is null, print all the keys and associated decryptions. Otherwise, the function should only print out potential plaintexts that contain that keyword.

   Note: you can implement additional functions.

c) **(5 points)** Show the output of your encrypt function (**part a**) on the following (key, plaintext) pairs:
   - k = 6 plaintext = "Get me a vanilla ice cream, make it a double."
   - k = 15 plaintext = "I don't much care for Leonard Cohen."

- k = 16 plaintext = "I like root beer floats."

d) **(5 points)** Show the output of your decrypt function (**part a**) on the following (key, ciphertext) pairs:
- k = 12 ciphertext = 'nduzs ftq buzq oazqe.'
- k = 3 ciphertext = "fdhvdu qhhgv wr orvh zhljkw."
- k = 20 ciphertext = "ufgihxm uly numnys."

e) **(5 points)** Show the output of your attack function (**part b**) on the following ciphertexts, if an optional keyword is specified, pass that to your attack function:
- ciphertext = 'gryy gurz gb tb gb nzoebfr puncry.' keyword = 'chapel'
- ciphertext = 'wziv kyv jyfk nyve kyv tpdsrcj tirjy.' keyword = 'cymbal'
- ciphertext = 'baeeq klwosjl osk s esf ozg cfwo lgg emuz.' no keyword

Problem 2 (10 points)

Before implementing the code of problem 3 for frequency attack, you will have a chance to perform the attack using our binary program.

There versions of binary programs are provided in the shared folder. One for MacOS (**Prob2_mac**), one for Ubuntu (**Prob2_ubuntu**) and one for SunOS (**Prob2_sun**). Use the binary program which is appropriate to your OS.

In the following, **Prob2_sun** is used for demonstration. You may need to gain execute access permission before being able to execute the binary files. To do this, use the following command
$ chmod +x <binary file>

The ciphertext to be decrypted is:

```
rnc qwr fv uwmrmgb eczwcr hceeqbce mgrcjjmbmajc rf rnfec unf qwc mg dfeeceemfg fv rnc kco qgx tgmgrcjjmbmajc rf qjj frncwe nqe accg ertxmcx vfw zcgrtwmce. rnc tecvtjgcee fv etzn hceeqbce, cedczmqjjo mg rmhc fv uqw, me faymfte
```

Assignments

1. (0 points) Decrypt this message using the provide program with the following instruction

a. Run the following command: **$ ./Prob2_sun ""** to compute a histogram of the incidence of each letter.
You can see letters ‘c’ is the most common cipher letter. So, we can guess that letter ‘e’ in the plaintext is substituted by letter ‘c’ or ‘e’ → ’c’. Moreover, notice that the three-letter word ‘rnc’ appears 3 times. With cipher letter ‘c’ corresponding to plaintext letter ‘e’, we can predict that ‘rnc’ is ‘the’, thus ‘t’ → ‘r’ and ‘h’ → ‘n’

b. Next, run the following command: **$ ./Prob2_sun __e__________h___t________**

The argument (called **subs**) **__e__________h___t________** represents 26 alphabetic character string where the character at position *i* is the substitution of *i*th character of the alphabet OR an underscore ‘_’ if the corresponding substitution is unknown. For example, from the previous guesses

- ‘e’ →’c’: ‘c’ is the 3rd letter in the alphabet. Thus, the letter at position 3 of the subs is ‘e’. Similarly, the letters at position 14 and 18 of the subs are ‘h’ and ‘t’, respectively.

After running the command, you should get:

the QWt FV UWMtMGB EeZWet HeEEQBeE MGteJJMBMAJe tF thFEe UhF QWe MG DFEEeEEMFG FV the KeO QGX TGMGteJJMBMAJe tF QJJ FtheWE hQE AeeG EtTXMeX VFW ZeGtTWMeE. the TEeVTJGeEE FV ETZh HeEEQBeE, eEDeZMQJJO MG tMHe FV UQW, ME FAYMFTE

The lowercase letters are our guessed plaintext letters. The uppercase letters are cipher letters. You can see two-letter word ‘tF’, so that ‘F’ is most probably ‘o’ (‘o’ → ‘f’). You can also see a three-letter word ‘UhF’, with ‘F’ being ‘o’, we can guess ‘w’ →’u’

Our subs now is: **__e__o_______h___t__w_____**

c. Run **$ ./Prob2_sun __e__o_______h___t__w_____**

You should get
the QWt oV wWMtMGB EeZWet HeEEQBeE MGteJJMBMAJe to thoEe who QWe MG DoEEeEEMoG oV the KeO QGX TGMGteJJMBMAJe to QJJ otheWE hQE AeeG EtTXMeX VoW ZeGtTWMeE. the TEeVTJGeEE oV ETZh HeEEQBeE, eEDeZMQJJO MG tMHe oV wQW, ME oAYMoTE

You see a three-letter word ‘thoEe’ and a six-letter word ‘otheWE’, thus ‘s’ →’e’ and ‘r’ →’w’. The we have ‘wQW’ with ‘W’ being ‘r’ then probably ‘a’ →’q’
Out subs now is: **__e_so_______h__at__w_r___**

d. Run **$ ./Prob2_sun __e_so_______h__at__w_r___**

The potential plaintext is
the art oV wrMtMGB seZret HessaBes MGteJJMBMAJe to those who are MG DossessMoG oV the KeO aGX TGMGteJJMBMAJe to aJJ others has AeeG stTXMeX Vor ZeGtTrMes. the TseVTJGess oV sTZh HessaBes, esDeZMaJJO MG tMHe oV war, Ms oAYMoTs

There is a six-letter word ‘seZret’ and four-letter word ‘sTZh’, so ‘c’→’z’ and ‘u’→’t’
We also notice that ‘HessaBes’ could be ‘massages’, so that ‘m’→’h’ and ‘g’→’b’
The new subs is : **_ge_so_m_____h__at_uw_r__c**

e. Run **$ ./Prob2_sun _ge_so_m_____h__at_uw_r__c**

The potential plaintext is
the art oV wrMtMGg secret messages MGteJJMgMAJe to those who are MG DossessMoG oV the KeO aGX uGMGteJJMgMAJe to aJJ others has AeeG stuXMeX Vor ceGturMes. the useVuJGess oV such messages, esDecMaJJO MG tMme oV war, Ms oAYMous

You see ‘aJJ’, so ‘l’→’j’. ‘useVuJGess’ becomes ‘useVulGess’ that probably is ‘usefulness’. Then ‘f’→’v’, ‘n’→’g’. You also see ‘tMme’ and ‘Ms’, so ‘i’→’m’.

The subs is now: **_ge_sonm_l__ih__at_uwfr__c**

f. Run **$ ./Prob2_sun _ge_sonm_l__ih__at_uwfr__c**
The potential plaintext is
the art of writing secret messages intelligiAle to those who are in Dossession of the KeO anX unintelligiAle to all others has Aeen stuXieX for centuries. the usefulness of such messages, esDeciallO in time of war, is oAYious

Now, it's easy to see 'b'→'a', 'p'→'d', 'y'→'o', 'k'→'k' , 'v'→'y' and 'd'→'x'
The subs is: **bgepsonm_lk_ihy_at_uwfrdvc**

g. Run **$ ./Prob2_sun bgepsonm_lk_ihy_at_uwfrdvc**

The plaintext is: *the art of writing secret messages intelligible to those who are in possession of the key and unintelligible to all others has been studied for centuries. the usefulness of such messages, especially in time of war, is obvious*

2. **(10 points)** What is the key of the above ciphertext?

Problem 3: (40 points)

In this problem, you are required to implement several functions useful to performing classical cipher attacks

a) **(10 points)** Implement a C function that performs **frequency attacks** on a mono-alphabetic substitution ciphers. This function should take a ciphertext string compute a histogram of the incidence each letter (ignoring all non-alphabet characters.) And return a list of pairs (letter, incidence percentage) sorted by incidence percentage.

b) **(15 points)** Implement a C function that takes a partial mono-alphabetic substitution (i.e., subs in Problem 2) and a ciphertext and returns a potential plaintext.
The partial mono-alphabetic substitution should be specified as follows:
As a 26-character string where the character at position i is the substitution of ith character of the alphabet, OR an underscore '_' if the corresponding substitution is unknown. The potential plaintext should be the ciphertext with values specified by the mono-alphabetic substitution replaced by the lower-case plaintext. If the corresponding character is unknown (i.e. '_' in the monoalphabetic substitution cipher) print the cipher text as an uppercase character.)

c) **(15 points)** Use your functions from (a) and (b) to decrypt the following cipher text:
"ztmn pxtne cfa peqef kecnp cjt tmn zcwsenp ontmjsw ztnws tf wsvp xtfwvfefw, c feb fcwvtf, xtfxevqea vf gvoenwk, cfa aeavxcwea wt wse rntrtpvwvtf wscw cgg lef cne xnecwea eymcg."

- Provided code: Prob3_skeleton.c
  + struct incidence_pair getIncidence(char *ciphertext): this function takes a ciphertext (string) and returns an incidence_pair structure (specified in the skeleton) which consist of an array of alphabetic letters and an array of letters' incidence percentage
  + char *monoalphabetic_substitution(char *ciphertext, char *subs): the partial mono-alphabetic substitution function that takes a ciphertext (string) and a subs (string).

Report Preparation

Report, including two parts: typed report and source code.

Content requirements

For typed report: The report should contain

(1) Problem 1: (1) proof of your code run actually (copy & paste of the console screen), and the proof of the cryptanalysis process.
(2) Problem 2: (1) proof of the decrypting process and analysis and (2) answer of question 2.
(3) Problem 3: proof of the decrypting process and *analysis*.

For source code: All source codes must be submitted.

Format requirements

Both parts must be compressed (zip) and submitted together

- Name the zipped file as <Year.Month.Date>.<Student_ID>.<LabXX>.zip
- For example: 2019.06.27.s1222222.Lab1.zip

Requirements for typed report

- Must be prepared using a Word processor (e.g. MS Word or Latex)
- Must be converted into PDF for submission
- Submitted individually
- Also name the file as <Year.Month.Date><Student_ID><LabXX-report>.PDF
- Example 2019.06.27.s122222.Lab1-report.pdf

Requirements for source code

- Also name the file as <Year.Month.Date><Student_ID><file_name>
- For <file_name>, use the name of the problem. For example, use "Prob1a" for problem 1a, "Prob1b" for problem 1b, and "Prob3" for problem 3. (Problem 2 does not need programing)
- Example 2019.06.27.s1222222.Prob1a.c, for source code file of Problem 1a.

Lab submission

- By email (send the zip file to d8202101, m5222108, and CC to pham). The subject of your mail should be: **[CN02]Lab1**
- Due date: by the mid-night of the last-session day (report preparation)
- Late submission is subject to penalty