SCIE1000 Project, Semester 1 2012

Topic 1: Cryptography

1 Introduction to Cryptography:

1.1 Science

Cryptography is the science of using mathematics to hide information. Cryptography allows us to store sensitive information, or to transmit it over insecure networks (such as the internet) so that it can only be read by the intended recipient. *Encryption* is the process of converting readable data (called the *plaintext*) into a form which hides its content, called the *ciphertext*. *Decryption* is the reverse process, with a ciphertext converted back into the corresponding plaintext.

A *cipher* is a mathematical function used in the encryption and decryption processes. Most ciphers use a *secret key* when encrypting, and different keys will typically encrypt a given plaintext into different ciphertexts. The key is usually only known by the person who encrypts the data, and the intended recipient. The secrecy of the key aims to ensure that even if an eavesdropper were to intercept the transmitted data, they would be unable to decrypt it. In general the security of encrypted data is dependent on two factors: the strength of the cipher, and the secrecy of the key.

The most common type of cipher is the *substitution cipher*, where we use the key to encrypt the plaintext one letter at a time, replacing each letter by another letter. In this project, we shall only consider plaintexts and ciphertexts written using the 26 lower-case English characters. We will ignore spaces and punctuation (since these can usually be determined from the context), and will assume that the plaintext is written in English.

The process of trying to derive the meaning of the ciphertext without knowing the key is known as *cryptanalysis*. This is often called "breaking" or "cracking" the cipher. The cipher we will consider in this assignment is the *General Mixed Alphabet* (GMA). The GMA is one of the more commonly used ciphers from history, due to its ease, simplicity and large number of keys. As we shall see in this project, however, it is relatively easy to crack this cipher.

A *General Mixed Alphabet* (GMA) has so many keys that it is not possible to check them all, even on very fast computers. In a GMA, plaintext is encrypted as follows.

- First a secret key is chosen. The key is a sequence (or table) of 26 letters, showing the ciphertext letter to which each plaintext letter is encrypted. Each letter of the alphabet must occur exactly once as a plaintext and once as a ciphertext in the key.
- The plaintext is encrypted one letter at a time. Each occurrence of the letter 'a' in the plaintext is encrypted to the first letter in the key, each occurrence of 'b' to the second letter in the key, and so on.

Example

We can encrypt Julius Caesar's famous saying "Veni, Vidi, Vici" using a GMA as follows.

- First, re-write the text to be enciphered to just use lower-case English characters. This is the plaintext: `venividivici`.
- Choose a key and use it to create the cipher table. The key is the second row of the following table.

Plain letter	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
Key: Cipher letter	s	a	h	m	u	z	e	b	i	n	v	c	d	j	o	w	r	f	k	p	x	t	g	l	q	y

- For each character in the plaintext, substitute it with its cipher equivalent. This is now the ciphertext: `tujitimitihi`.

Note that we do not re-insert spaces, punctuation or capitals into the ciphertext, as they could make the ciphertext easier to crack.

When decrypting the message, we perform the reverse procedure: we take each character in the ciphertext and find its plaintext equivalent, using the cipher table. For example, if the ciphertext "stuhsuksf" had been encrypted using the above key, then the corresponding plaintext is "avecaesar", or "Ave, Caesar" with spaces and punctuation inserted.

Frequency Analysis

Despite the enormous number of keys in a GMA, it has been proven to be insecure: as early as the 9th century, the Arab scholar Al-Kindi was using *frequency analysis* to crack this type of cipher. The basis of frequency analysis is that — no matter which language is being used — not all letters occur equally often. In English text, for example, the most common letter is typically 'e' (which makes up approximately 12.702% of all letters) and the least common letter is 'z' (with a frequency of 0.074%). Compare this to a frequency of $1/26 \approx 3.85\%$, which is what we would expect if each letter had the same frequency. See Figure 1 for the relative frequencies of characters in English.

Letter	Frequency	Letter	Frequency
e	12.702%	**m**	2.406%
t	9.056%	**w**	2.360%
a	8.167%	**f**	2.228%
o	7.507%	**g**	2.015%
i	6.966%	**y**	1.974%
n	6.749%	**p**	1.929%
s	6.327%	**b**	1.492%
h	6.094%	**v**	0.978%
r	5.987%	**k**	0.772%
d	4.253%	**j**	0.153%
l	4.025%	**x**	0.150%
c	2.782%	**q**	0.095%
u	2.758%	**z**	0.074%

Figure 1: Relative frequencies of English characters, in descending order of frequency.

Thus, the basis of frequency analysis is that if we find the frequency of each letter in the ciphertext, the most common letter probably corresponds to the letter 'e', the second most common letter corresponds to 't', and so on. Note that it is usually not this simple, as the frequencies in Table 1 are *averages* for text written in English; specific texts will probably have different frequencies. For example, the French author Georges Perec wrote the 200-page novel *La Disparation* in 1969 without using a single letter 'e' (which is a common letter in French as well); Gilbert Adair then translated this into English (entitled *A Void*), again avoiding usage of 'e'. As such, you might need to experiment with the top few most common letters in different orders. Frequency analysis typically works better on longer pieces of ciphertext, where the small variations do not affect the overall frequencies as much (for example, the most common letter in "Veni, Vidi, Vici" is 'i', followed by 'v').

In addition to counting the frequencies of single letters, it may also be useful to consider sequences of two or three consecutive letters, called *digrams* and *trigrams*, respectively. The 30 most common English digrams are (in decreasing order) TH, HE, IN, ER, AN, RE, ED, ON, ES, ST, EN, AT, TO, NT, HA, ND, OU, EA, NG, AS, OR, TI, IS, ET, IT, AR, TE, SE, HI and OF. The twelve most common trigrams are (in decreasing order) THE, ING, AND, HER, ERE, ENT, THA, NTH, WAS, ETH, FOR and DTH. Combining individual frequency counts with the frequencies of common digrams and trigrams is a powerful technique for breaking a GMA.

1.2 Mathematics

Because each letter of the alphabet occurs exactly once in the key for a GMA, this key is an *ordered arrangement* of the alphabet. Mathematically, this is called a *permutation* of the alphabet. Given n items, there are $n!$ (pronounced "n-factorial") distinct permutations of those items.

1.3 Python

To plot a bar graph (or histogram) where the x value for each bar is in the array `bins` and the height of each bar is in the array `heights`, use the following commands:

```
figure()
bar(bins, counts)
show()
```

1.4 Questions:

0. (0 marks) A number of the following questions vary between students. Let T be the two digit number corresponding to the **last two digits of your student number**, so T is a number between 00 and 99 (inclusive). For example, if your number were 42136702, then $T = 02$. (It is important that you use the value of T that corresponds to your student number; if you do not, you will lose a large number of marks. If you have any questions, please ask a tutor.)

1. You must complete an online ethics module before doing any work on this project. Your completion of the module will be automatically recorded; **if you do not complete the module prior to handing in the initial submission, then you will lose half of the marks you would otherwise have obtained.**

   (a) (0 marks) Complete the module as follows:

   * ∗ Open a window using Firefox (it is the most stable browser for this).
   * ∗ Go to `https://scenarios.sblinteractive.org/v2/uq/loginform.aspx`
   * ∗ Log in to the system using your UQ login and password.
   * ∗ Use the Search box to search for: SCIE1000
   * ∗ To start the module, click on "Start Scenario".
   * ∗ Work through the module.

   (b) (5 marks) in about 150 words, summarise what you learned about ethics and academic integrity in the context of completing your project, and collaboration when doing mathematics and computer programming.

2. (3 marks) The Blackboard site contains files "GMA_plain_1.txt" and "GMA_keys_1.txt". These files contain (respectively) 100 pieces of plaintext and 100 keys for the GMA, numbered from 00 to 99. Use the plaintext/key with number T, where T is the value you calculated in Question 0. Encrypt the plaintext.

3. (3 marks) The Blackboard site contains files "GMA_cipher_2.txt" and "GMA_keys_2.txt". These files contain (respectively) 100 pieces of ciphertext and 100 keys for the GMA, numbered from 00 to 99. Each piece of ciphertext has been encrypted using the corresponding key. Use the ciphertext/key with number T, where T is the value you calculated in Question 0. Decrypt the ciphertext. Your answer must include the plaintext, with appropriate spaces and punctuation inserted.

4. **(a)** (2 marks) The key for a GMA is a permutation of the 26 letters, which is hard to remember. In practice, GMAs typically employ a secret keyword, as follows.

∗ Write the secret word across the page, but not repeating any letter that has already been written.

∗ Working from the start of the alphabet, under the keyword, write each letter that has not yet been written on the page. If you reach the end of the keyword, start writing on a new line.

∗ You should now have all 26 letters written exactly once each. Read these letters column by column; this forms the key for the GMA.

Find the GMA key that corresponds to your family name. Show all working.

(b) (2 marks) Using the key from Part (b), encrypt the phrase "Veni, Vidi, Vici".

5. (5 marks) The Blackboard site contains file "GMA_plain_3.txt". This file contains 100 pieces of plaintext, numbered from 00 to 99. Use the plaintext with number T, where T is the value you calculated in Question 0. Find the absolute and relative frequencies of each letter within the plaintext. (Note: absolute frequency is the count, and relative frequency is the count divided by the total length of the text.)

6. **(a)** (0 marks) Write a Python program that does the following. (Much of the program doesn't do anything useful yet; we will modify this later on.)

- Include a new function called `isBig` which has two values passed in, called `relFreq` and `thresh`. This function must return `True` if the value of `refFreq` is greater than or equal to the value of `thresh`. Otherwise, the function must return `False`.
- Ask the user to select their choice from four initial options: 1 = encryption, 2 = decryption, 3 = frequency analysis or 4 = count repeats.
- Irrespective of which option is chosen, read some text from a specified file name. (The hints below will help you to do this.)
- If the user chooses Options 1 or 2:
  - · Read a 26-letter key from a file and then print the key. (Again, see the hints below.)
  - · Print the text that was read from the file.
- If the user chooses Option 3, include the following lines of code:

```
bigFreq = input("What relative frequency is 'large' (as a %)? ")
print isBig(2, bigFreq)
print isBig(10, bigFreq)
```

- If the user chooses Option 4, print the message "`Counting repeats is not implemented yet.`".
- Has variables with meaningful names, and has appropriate comments.

Hint(s):

- You may assume that all input values are valid (for example, the user will select a valid Option from 1 to 4).
- The SCIE1000 Blackboard site contains a file called "cryptoIO.py" which contains some useful commands. To use these commands, place a copy of that file in the same folder as the programs that you are writing, and import it into your program using the command

```
from cryptoIO import *
```

after the other import statements. For this question, the following commands which are defined in "cryptoIO.py" may be useful:

`readFile:` This command reads the name of a file from the keyboard, then reads all of the text in that file into an array.

`readKey:` This command reads the name of a file from the keyboard, and then reads a GMA key from the file. The key should be a sequence of 26 letters, with each letter occurring exactly once. The first letter is the encryption of 'a', the second is the encryption of 'b', and so on.

- As an example, here is a sample program using those two commands:

  ```
  from __future__ import division
  from pylab import *
  from cryptoIO import *

  text = readFile()
  print "The text is:"
  print text

  key = readKey()
  print "The key is:"
  print key
  ```

- Note that you must **not** change the contents of cryptoIO.py under *any* circumstances. We suggest that you **do not** read the code in cryptoIO.py, as it uses some programming constructs that you have not learned.

(b) (8 marks) Print and submit a copy of your program from Part (a). Marks will be awarded for:

  $\ast$ Adherence to the program specifications given above

  $\ast$ Appropriate programming style, structure and logic

  $\ast$ Appropriate print statements, with helpful text explanations of the output

  $\ast$ Use of comments and meaningful variable names

(c) The Blackboard site contains files "GMA_plain_1.txt" and "GMA_keys_1.txt". These files contain (respectively) 100 pieces of plaintext and 100 keys for the GMA, numbered from 00 to 99. Use the plaintext/key with number T, where T is the value you calculated in Question 0. Save the correct plaintext to a file called "test1plain.txt", and the key to a file called "test1key.txt". (For the key, save only the 26 letters that appear after "Cipher:".)

Test your program in the following ways. (You must include a printed copy of the output from your program in your submission.)

  (i) (1 mark) Select Option 2, and use the text file and key file you just created.

  (ii) (1 mark) Select Option 3, and enter the value 5 for the 'large' relative frequency.

All questions above this line must be completed as part of your initial project submission.

---

7. **(a)** (0 marks) Modify your Python program from Question 6 so that:

- If the user selects Option 1, then the program encrypts the text using the given key, and prints the encrypted text.
- If the user selects Option 2, then the program decrypts the text using the given key, and prints the decrypted text.
- If the user selects Option 3, then the program:
  - · **no longer** prints the output from the two calls to the function `isBig` (so you should delete those two lines of code).
  - · plots a histogram of the absolute frequencies of the letters in the text (you may use the numbers 1 to 26 on the `x` axis, rather than letters).
  - · prints each letter which has a **relative frequency** greater than or equal to the specified threshold value, along with the relative frequency. You **must** use the function `isBig` in your program as part of this.

- If the user selects Option 4, then the program:
  - · reads a string of letters from the keyboard (see the hint below)
  - · prints the string of letters and the frequency with which that string occurs in the text file.
- has variables with meaningful names, and is appropriately commented.

Hint(s):

The file "cryptoIO.py" also contains the following two commands that may be useful.

`readString`: This command reads some text from the keyboard.

`stringHere`: This command takes an array of text, and the index of an entry in that array (starting at 0), and another array of text, and returns True if the second text string occurs starting at that index in the first text string, and False otherwise.

- As an example, here is a sample program using those two commands:

```
from __future__ import division
from pylab import *
from cryptoIO import *

text = readFile()
Str = readString()
print "The typed text is", Str
if stringHere(text, 0, Str):
    print "The word", Str," occurs at the start of the given text."
```

(b) (15 marks) Print and submit a copy of your program. Marks will be awarded for:

∗ Adherence to the program specifications given above
∗ Appropriate programming style, structure and logic
∗ Appropriate print statements, with helpful text explanations of the output
∗ Use of comments and meaningful variable names

(c) Test your program in the following ways. (You must include a printed copy of the output from your program in your submission; in each case, verify the output against your hand calculations.)

(i) (2 marks) Use your program to repeat Question 3.

(ii) (2 marks) Use your program to repeat Question 4.

(iii) (2 marks) Use your program to repeat Question 5.

(iv) (3 marks) The Blackboard site contains a file called "phil.txt", which is a copy of the philosophy section from the SCIE1000 notes. Use your program to find the number of occurrences of the words "math", "science" and "the" in that file.

(d) (3 marks) Write a brief user guide which explains how to use the program. (Your user guide should **not** assume that the user has read this assignment question sheet.) The guide should contain all necessary information about:

- What the program does.
- What input is requested by the program, and what valid input it can take.
- What output the program gives.
- Any assumptions you have made, or any special cases.

This is a user guide, not a programmer's manual. Do not describe the algorithm you have used or internal details of the program. Instead, if someone with a basic understanding of computers (and a good understanding of the science relevant to your project topic) wanted to run your program, what would they need to know?

8. **(a)** (3 marks) Find a large piece of electronic text (around 10000 or more words, on the web), save it to a file, and use your program to perform a frequency analysis on it. Submit a reference to the text (for example, the web address), the frequency histogram and the list of all letters with a relative frequency of 7% or more.

(b) (5 marks) Figure 1 presents "standard" relative frequencies of letters in English text. Compare the results you obtained in Part (a) with the data in Figure 1. In about 150 words, discuss similarities and differences, and explain why your results are similar to, or different from, the data in Figure 1.

9. (25 marks) When answering this question, if relevant you may include graphs, diagrams, tables of values, equations or calculations (which will not be included in the word limit), but your response should be predominantly text-based. The report must be typed, written in a professional style, and be within 10% of 2000 words in length. Marks will be deducted for any report with length outside this range.

It is **not** appropriate to use other sources or references in your report; all of the discussion and recommendations should arise solely from your own work, including output from your program. The course Blackboard site contains a Criteria Sheet for this report, showing how marks will be allocated.

A secretive political lobby group (SPLG) employs you to investigate methods by which they can keep their communications secret. They have heard that the GMA has many possible keys, is simple to understand, and is easy to use by hand, so they would like to use it to encrypt their communications. They understand that the cipher can be broken, but they want some evidence about how easy or difficult that is. They pose the following questions:

(a) How hard is it to crack some ciphertext encrypted using a GMA and an unknown key?

(b) Is there anything that can be "done" to a plaintext message that makes the corresponding ciphertext harder to crack?

Write a report for SPLG. The report must meet the following criteria:

– It must be understandable by SPLG's management, who have some knowledge of simple mathematics, but are not experts.

– It should **not** describe any of the mathematical details of the cipher, or anything about the programming you did; instead, SPLG wants you to answer their specific questions.

– It must include four sections:

1. An "Executive Summary" of around 150 words, giving a brief overview of your findings, and listing the answers to their questions. The rest of the document will expand on the content of the executive summary.
2. A detailed description of **your actual** experiences when attempting to crack ciphertext encrypted using a GMA and an unknown key.
   The Blackboard site contains a file "GMA_cipher_big.txt". The file contains 100 pieces of ciphertext, each encrypted using a GMA with a secret key. Use the ciphertext with number T, where T is the value you calculated in Question 0. Decrypt the ciphertext. Your answer should include:
   · frequency information about letters in the text, from your program
   · frequency information about some common repeated strings of letters in the text, and the possible plaintext letters to which they correspond
   · a detailed description of what you did, each step you followed, which letters you decrypted in which order, how you used information about repeated strings, which choices worked and which didn't work (this discussion will probably be hundreds of words long). A large number of marks will be assigned to this discussion.
   · the key.

  · the plaintext, with spaces and punctuation inserted.

  If you cannot crack the cipher, it is still possible to receive full marks if your discussion is very good, and you explain why your approach didn't work.

3. Assume that SPLG needs to transmit a secret message about a special meeting they are holding. The message needs to be about 300 **characters** long, and it **has** to be encrypted with the GMA cipher.
   (a) Discuss some techniques which could be used to make it harder for the "enemy" to crack the message. (Note: all that you can control is the choice of key, and the content of the message.)
   (b) Use the principles you outlined in Part (a) to write a plaintext message (remember, it is about a political meeting, and should be about 300 characters long) and a key, and explain why the message would be 'hard' to crack.
4. A "Conclusions/answers" section containing brief answers/responses to their questions, based on the data and experiences you presented above.

The end