NWEN 243 Lab 1

Part 1

Experience thinking about security

- Write a small program to encode a block of plain-text using a modified Caeser cypher.
- Write a small program to decode a block of cypher-text using a modified Caeser cypher.

Requirements

- This lab an individual lab written in C.
- We will be writing programs that you execute from the shell command line.
- You are to write two small C programs that will
    1. encode using your key, and
    2. decode using the same key –

  using **a modified Caeser cypher**.

For example

```
%>cat file.txt | encode password | decode password > newfile.txt
// I should then be able to execute diff, and see no changes, e.g.:

%> diff -i file.txt newfile.txt
```

The Modified Cypher

The modified Caeser you will be implementing is a combination of Caeser cypher I and II.

Caeser cypher I used an offset alphabet, where the mapping between plain and enciphered characters was determined by a numerical offset, in the following example 13, so A maps to M:

Plain	A	B	C	D	E	F	G	H	…
Enciphered	M	N	O	P	Q	R	S	T	…

The Caeser cypher II uses a 'key word' where it is placed at the start of the encipher mapping – sans duplicates, the key in this example is "I came, I saw" (without punctuation or whitespace etc.):

Plain	A	B	C	D	E	F	G	H	…
Enciphered	I	C	A	M	E	S	W	X	…

The next letter in the encipher mapping is the next non duplicate letter of the alphabet – in this case X follows W, and it is not a duplicate. This is followed by Y, Z, A, B, … Note A is not used as it is a duplicate.

In our modified cypher, what we will do is use an offset (like cypher I) which will be the length of the key (including duplicates and whitespace and punctuation, i.e., the key "I came, I saw" has a total length of 13. From this offset of 13, we will then apply a normal Caeser cypher II, as in the following example:

1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6
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
H	J	K	L	N	O	P	Q	R	T	U	V	I	C	A	M	E	S	W	X	Y	Z	B	D	F	G

Note: numbers included only for your reference

So, 'Hello' – would encode to: 'QNVVA'

Assumptions
You may assume text is in ASCII and that you do not need to preserve case (just make it all upper case – note '-i' option to diff above) encode symbols, punctuation, spaces or whitespace in general (i.e just pass it through without change), e.g.: "don't do it" becomes "efh'w ef xw"

Skeleton Code
A small skeleton will be provided on the course web page to ease you into this project. No IDE is provided, the command line and any editor (vi, emacs, kwrite) will be sufficient. You can compile your code using the command line:

```
%> gcc encode.c -o encode
```

Part 2

Experience thinking about crypto-analysis
- Write a program to perform frequency analysis of a block of cypher-text
- Determine (using bruteforce) how many keys were used in the polyalphabetic Caeser cypher

Requirements
- This lab an individual lab written in C.
- We will be writing programs that you execute from the shell command line.
- You are to complete a C program that will allow you to determine the number of keys used in a polyalphabetic cypher, once this has been discovered – the problem becomes one of simple frequency analysis. As we have powerful computers, this is easy to brute force. The idea is to simply try different numbers of keys until we find one that gives a 'mostly' decoded result. That will give us the number of keys. For example, if two keys are used, then all the even numbered characters in the text are in one sub-cyphertext, while all the odd numbered characters are in the other sub-cyphertext. Three keys, would mean the cyphertext split 3 ways, etc.

    3. For each possible number of keys, i
    4. Split the cyphertext into i sub-cyphertexts and apply frequency analysis to each of them
    5. Recomine the sub-cyphertexts the entire 'partly decoded' text, for each nominated number of keys and output the result.

For example:

```
%>crack 4 test
```

This will try 1 through 4 keys

Assumptions
You may assume text is in ASCII and that you do not need to preserve case (just make it all upper case) don't encode symbols, punctuation, spaces or whitespace in general (i.e just pass it through without change), e.g.: "efh'w ef xw" becomes "don't do it"

Skeleton Code
A small skeleton will be provided on the course web page to ease you into this project. No IDE is provided, the command line and any editor (vi, emacs, kwrite) will be sufficient. You can compile your code using the command line:

```
%> gcc crack.c -o crack
```

Do not use your own name!

Submission & Marking

- This project will be marked by your tutors using an automatic marking script. So, ensure you have followed the instructions exactly. It also MUST run on our systems, ensure all needed files are included, and that it compiles on embassy.
- Include a pdf document outlining each of your functions, the algorithms, and why you did it that way. Is it efficient, robust etc. How did you test it.

- Marking (total of 10):
    - The Document is worth 2 marks
    - The WORKING code for part 1 is worh 3 marks, while the working code for part 2 is work 5 marks – code that compiles, but fails some or all tests will be awarded partial marks. Code that fails to compile, will receive no marks for the code section, so ensure your code compiles, even if it isn't right – the tutor will look at it personally.

Any problems, remember to ask your tutors and use the forum outside your lab.