Project 1

Grading and Submission Policies: Project 1 will contribute to the class grade as specified in the syllabus. All students must submit for Project 1 using the appropriate link in the Tests & Quizzes content area, and using the submission naming convention specified below. If you want just one group member to submit your long submission, the other team member(s) have to submit some text or file with a pointer to the submitting student in their team (or otherwise I cannot assign a score).

The project should be realized by a team of 2, 3 or 4 students (only well motivated exceptions will be considered acceptable). The implementation can be in any programming language. I recommend C or C++ for speed optimization (alternatively Python), as these are the most recommended (for combined performance and user convenience) programming languages to implement cryptography solutions in the real world. The project comes with a minimal assignment and requires a submission of both software and a report being graded mainly by the TAs according to scoring criteria defined below; any additional work you perform will be considered extra credit work (specifically, Extra Credit 3) if later also submitted under the appropriate space for Extra Credit 3 under Tests and Quizzes. Teams are supposed to split the amount of work more or less equally among the team members; if a team splits the work in a way that is too unbalanced, details should be mentioned to the instructor and the score given to team members may be suitably unbalanced.

Project 1 (Cryptanalysis of a class of ciphers):

This cryptanalysis project consists of a software implementation of an algorithm that tries to decrypt an L-symbol challenge ciphertext computed using a specific cipher. Informally speaking, your program's goal is to find the plaintext used to compute this ciphertext within a reasonable amount of time. Specifically, your program should print on screen something like "Enter the ciphertext:", obtain the ciphertext from stdin, apply some cryptanalysis strategy and output on screen something like "My plaintext guess is:" followed by the plaintext found by your strategy. In doing that, your program is allowed access to:

1. The ciphertext (to be taken as input from stdin)
2. A plaintext dictionary (to be posted on top of this web page), containing a number q of plaintexts, each one obtained as a sequence of space-separated words from the English dictionary
3. Partial knowledge of the encryption algorithm used (to be described below).

Your program is not allowed access to:

1. The key used by the permutation cipher.
2. Part of the encryption scheme (to be detailed below).

The plaintext is a space-separated sequence of words from the English dictionary (the sentence may not be meaningful, for sake of simplicity). The key is a map from each English alphabet (lower-case) letter (including <space>) to a list of numbers randomly chosen between 0 and 105, where the length of this list is the (rounded) letter's frequency in English text, as defined in the table below. The ciphertext looks like a sequence of comma-separated numbers between 0 and 105, obtained as a sequence of encryptions of words, where each word is encrypted as a comma-separated list of numbers between 0 and 105, and these numbers are computed using the table below.

English letters	Average frequency	Key values (randomly chosen distinct numbers between 0 and 105)
<space>	19	k(<space>,1),…,k(<space>,19)
a	7	k(a,1),…,k(a,7)
b	1	k(b,1)
c	2	k(c,1),k(c,2)
d	4	k(d,1),…,k(d,4)
e	10	k(e,1),…,k(e,10)
f	2	…
g	2	
h	5	
i	6	
j	1	
k	1	
l	3	
m	2	
n	6	
o	6	
p	2	
q	1	
r	5	
s	5	
t	7	
u	2	
v	1	
w	2	
x	1	
y	2	
z	1	

The cipher works as follows. It takes as input a plaintext from a message space and a key randomly chosen from a key space and returns a ciphertext.

- The message space is the set {<space>,a,..,z}^L. In other words the message m can be written as m[1]...m[L], where each m[i] belongs to set {(space>,a,..,z}
- The ciphertext c can be written as c[1],...,c[L], where each c[i] belongs to set {0,..,105}. To avoid ambiguities, cyphertext symbols are separated by a comma.
- The key space is the set of random maps from {0,..,26} to a permutation of all numbers in {0, …,105}, grouped in 27 lists, each list having length determined by column 2 of the table below.
- The encryption algorithm works as follows. For each message character m[j], the algorithm finds m[j] in column 1 of the table below, and returns one of the keys in column 3 of the same row. The computation of which key is returned by the algorithm is based on a scheduling algorithm which is intentionally left unknown and is a deterministic algorithm (that is, it does not use new random bits) that may depend on j, L and the length of the list on that row.
- The decryption algorithm does the inverse process. On input a ciphertext character, it finds the ciphertext character in column 3 of the table, and returns the column 1 plaintext letter (possibly including <space>) that is on the same row.

For instance, assume k(<space>,1)=76,...,k(<space>,19)=94, k(b,1)=23, k(c,1)=11, k(c,2)=98, k(g,1)=34, k(g,2)=56. Then the plaintext “cbcb gbgg gcb” may be encrypted as “98,23,11,23,79,34,23,56,34,82,34,11,23”. (Just for simplicity, in this example the plaintext was not a sequence of English words.)

Your program will be scored based on two tests.

In the first test, your program will be run many times, each time on a new ciphertext, computed using the above encryption scheme and a plaintext randomly chosen from the plaintext dictionary, with a different scheduling algorithm. On the first execution, the scheduling algorithm will compute “j mod length(list)” and use this result to select the element of that position in the list. On the other executions, the scheduling algorithms will be more and more complex variations of this one. In this test we will likely choose L=500, and a plaintext dictionary with q=5 plaintexts.

In the second test, your program will be run a few times, each time on a new ciphertext, computed using a plaintext obtained as a space-separate sequence of words that are randomly chosen from a subset of the set of all English words (specifically, a few words taken from the attachment english_words.txt at the top of this page, to be published soon) and the above encryption scheme, with a different scheduling algorithm. In this test we will likely choose L=500.

Your executable file should be named "<last name1>-<last name2>-<last name3>-decrypt". Upon execution, it should obtain the ciphertext from stdin, and finally return the output plaintext on stdout within x minutes (or else it will be declared to default to an incorrect guess); most likely, we will choose x = 1 on test 1 and x = 3 on test 2.

If you want to propose more than one cryptanalysis approach, you need to clarify that in your report, and each of your approaches will be tested. You cannot pick more than one approach per team member. Your overall team score will be an average across the success of the various approaches

approaches

approaches.

Your accompanying **report** should at least include the following sections:

1. title of your project (based on your approach); something like "Cryptanalysis of a class of ciphers based on (...)"; the symbol (...) should be replaced with an expression summarizing the main idea(s) in your approach

2. an introduction containing the team member names; the list of project tasks performed by each student in the team; the number of cryptanalysis approaches you are submitting; and any modifications you made with respect to the above specifications

3. a detailed informal explanation (using much more English than pseudo-code) of the cryptanalysis approach or approaches used in your program

4. a detailed rigorous description (using much more pseudo-code than English) of the cryptanalysis approach or approaches used in your program

Allowed extensions (to be considered as **extra credit**) include any one among the following:

1. a report section containing a brief (i.e., <= 2 pages) survey on permutation ciphers
2. a report section containing a brief (i.e., <= 2 pages) survey on cryptanalysis approaches for permutation ciphers
3. anything else you want to add

Your submission should be a zip file containing at least the following files: project report (in pdf form), source and executable. Please name your zip file as <last-name1><last-name2><last-name3>-cs6903f20project1 and your contained files as <last-name1><last-name2><last-name3>-report, <last-name1><last-name2><last-name3>-source, <last-name1><last-name2><last-name3>-decrypt. Your submission will be judged based on the following **grading criteria**:

1. software correctness and usability (i.e., if you followed all of the above instructions, if software runs correctly, and is easy to use)
2. quality of report (i.e., how well written is your report)
3. cryptanalysis success (i.e., how many challenge ciphertexts your program successfully decrypted). If there are two or more submissions successfully decrypting the same number of challenge ciphertexts, we may rank them based on their (faster to slower) running time taken to produce their outputs.

A good cryptanalysis strategy not guessing any plaintext or an uninteresting cryptanalysis strategy guessing all plaintexts in the first test will be rewarded with a score around the B or B+ level. A good cryptanalysis strategy guessing all plaintexts in the first test will be rewarded with a score around the A- level. Mild success in the second test should be enough for a score in the A level.

The top team(s) will be rewarded with extra credit.

Due date is on the syllabus. No late submissions can be accepted without score penalty and early submissions are encouraged. You are strongly recommended to submit any questions to the TAs (see Syllabus content area for their contact info) and the instructor.