6.857 Homework	**Problem Set 1**	# 1-2 - Reused Pad Cipher
Erica Du		
Skanda Koppula		
Jessica Wang		February 22, 2015

A. They were encrypted using the same pad. The two words are ADVERSARIAL AND MATHEMATICS. We used the following script:

```
CT1_str = 'd2 6b a5 0d 27 6a 34 2d 8e 53 0e'
CT2_str = 'de 6e a7 00 30 74 34 2b 8e 51 11'

CT1 = [int(byte, 16) for byte in CT1_str.split(' ')]
CT2 = [int(byte, 16) for byte in CT2_str.split(' ')]

def xor(xs, ys):
    '''Perform pairwise XOR operation on two lists'''
    return [x ^ y for x, y in zip(xs, ys)]

X = xor(CT1, CT2)

dictionary = open('dictionary', 'r').read().split()
words = set()
for w in dictionary:
    if len(w) == len(X):
        try:
            words.add(w.upper().encode())
        except:
            print 'Problem encoding', w

for PT1_str in words:
    PT1 = [ord(byte) for byte in PT1_str]
    PT2 = xor(PT1, X)
    PT2_str = "".join([chr(byte) for byte in PT2])
    if PT2_str in words:
        pad = xor(PT1, CT1)
        print('PT1 = %s, PT2 = %s, pad = %s' % (PT1_str, PT2_str, pad))
```

B. There are four messages and three unique pads. Therefore at least two messages will have the same two pads applied to them ($\binom{3}{2} = 3$). If we XOR those two pads together, the pads will cancel. Once we obtain the pad-canceled XOR of two messages, we peel apart the two messages via crib-dragging.

First, to identify which of the six possible pairwise combinations have the pads cancel out, we compute all six and count the frequency of '0' in the hex representation of each cross. All lowercase ascii [a-z] share the same last four bits, so the XOR of two plaintext paragraphs will contain a signature with plenty of zeroes.

Now we try peeling apart the two messages via crib-dragging. We took a few common words like ' the ' or ' is ' and XOR'd them along every position in our message product. If the common word is in that position in one of the messages, bits of the other message will be revealed. We extrapolate and guess words from those bits, and repeat the process. `grep` us helped extrapolate possibilities for words given the dictionary. We were also thinking of XOR'ing big words and testing for resulting smaller words in the other message.

We wrote the following interactive script to speed up and automate entering a crib, XOR'ing along the positions, and choosing the correct position (if any).

```
import itertools
import sys
import re

def read_ciphertexts(filename):
    '''Reads ciphertexts from file and removes spaces'''
    ciphers = open(filename, 'r').read()
    ciphers = ciphers.split('\n')[::2]
    return [text.replace(' ', '').decode('hex') for text in ciphers]

def get_dictionary(filename):
    '''Read dictionary file and returns set of words'''
    dictionary = open(filename).read()
    return set(ciphers.split('\n'))

def is_int(i):
    '''Tests if variable is valid integer value'''
    try:
        int(i)
        return True
    except ValueError:
        return False

def str_xor(str1, str2):
    '''Perform pairwise XOR operation on two strings'''
    return ''.join(chr(ord(x) ^ ord(y)) for x,y in zip(str1,str2))

def get_cancelled_pad(ciphertexts):
    '''Performs pairwise XOR's between ciphertexts
    and determines which cross has the pad canceled out
    by frequency of zeroes'''

    best_i, best_j, best_freq = 0, 0, 0
    for i, text1 in enumerate(ciphertexts):
        for j, text2 in enumerate(ciphertexts):
            if i == j: continue
            count = sum([1 if c == '0' else 0 for c in str_xor(text1, text2).encode('hex')])
            if count > best_freq:
                best_i, best_j, best_freq = i, j, count
```

```
    print 'Using ciphertext cross with', best_i, 'and', best_j
    return str_xor(ciphertexts[best_i], ciphertexts[best_j])

def peel_apart_messages(canceled_pad, charset):
    '''Interactive function for user to guess crib repeatedly
        and make it easy to peel apart two messages'''

    message1 = '_'*len(canceled_pad)
    message2 = '_'*len(canceled_pad)

    while True:
        print 'Current Message 1:'
        print message1
        print 'Current Message 2:'
        print message2

        crib = raw_input("Enter crib: ")

        num_positions = len(canceled_pad) - len(crib) + 1
        results = [0]*num_positions
        for i in xrange(num_positions):
            pos_result = str_xor(canceled_pad[i:i+len(crib)], crib)
            results[i] = pos_result
            if re.search(charset, pos_result):
                sys.stdout.write(str(i)+': '+pos_result+'*''\t\t')
            else:
                sys.stdout.write(str(i)+': '+pos_result + '\t\t')
            if i%3 == 0: sys.stdout.write('\n')
        sys.stdout.write('\n')

        response = raw_input("Enter the position, 'n' for no match, or 'q' to quit: ")
        if response == 'none' or response == 'n':
            print 'No changes made'
        elif is_int(response) and int(response) < num_positions:
            response = int(response)
            one_or_two = raw_input("Is crib part of message 1 or 2? '1' or '2': ")
            if one_or_two == '1':
                message1 = message1[:response] + crib + message1[response+len(crib):]
                print results[response]
                message2 = message2[:response] + results[response] + message2[response+len(cri
            elif one_or_two == '2':
                message2 = message2[:response] + crib + message2[response+len(crib):]
                message1 = message1[:response] + results[response] + message1[response+len(cri
            else:
                print 'Invalid input!'
        elif response != 'quit' or response != 'q':
            break
        else:
            print 'Invalid input'
```

```
ciphertexts = read_ciphertexts('ciphers.txt')
canceled_pad = get_cancelled_pad(ciphertexts)

charset = '^[a-zA-Z0-9.,?! :;\'"]+$'
peel_apart_messages(canceled_pad, charset)
```

From this we discovered that we were able to decrypt parts of the **first** and the fourth messages. Once the latter portions of the two messages were decrypted (by guessing common cribs such as ' the ' and ' is ', and daisy-chaining together the cribs that those revealed), we were reminded of a couple of quotes from the movie, The Imitation Game, which ultimately led us to find the two messages. The two messages:

```
Current Message 1:
I'm designing a machine,that will allow us to break every message, every day, instantly   now

Current Message 4:
I like solving problems  Commander. And Enigma is the most difficult problem in the world now
```