Programming Assignment #2

Spring 2019

Due: Sunday, March 3, *before* 11:59 PM

Abstract

In this programming assignment, you will be responsible for determining a cryptographic key that is used to encrypt the 10 cipher texts provided. In order to complete this programming assignment, you need to understand the problems inherent to the use of a single key to encrypt multiple messages.

Attachments

Project Description

Deliverable

Key.txt

Decrypted_messages.txt

Project-2-Code--<Lastname>-<Firstname>.(c)(py)(java)

(Note! Capitalization and spelling of your filename matters!)

(This project has been inspired and adopted from Dan Boneh projects)

1.0 Submission Requirements

1.1 Preface

As when responding to a professional work solicitation/submission, we gain practice with the protocol that Submission

Requirements must be met for the submission to be considered for credit.

Most companies set this standard, and most government agencies uphold such requirements per their legal contracting and acquisition processes. So we learn these conventions in our course projects and gain practice with formal “Project Solicitation”-style Task Definitions.

1.2 Submission Mechanism

Please upload the submissions identified below to CANVAS using the Assignments tab. No other form of submission can be accepted.

1.3 Program Code file

For your program code, submit a single file with .c or .cpp or.java extension that provides all parts of the project as a single program. The required file naming convention is

Project-2-Code--<LastName>-<FirstName>

are replaced with your last name and first name.

1.4 Incremental Submission

Upload your submissions as you progress in creating results, which will overwrite previous partial submissions. Only the last submission received before the due date/time will be graded.

2.0 Reference Literature

2.1 Lecture Slides

CSI 4415Khoshavi’s Modules which are available in CANVAS.

2.2 Textbook

The resources have been identified in the Course Syllabus.

3.0 Project Functionality

3.1 Overview

The objective of the project is to demonstrate, from an attackers perspective, how insecure the use of a single key is over multiple messages. Your goal is to develop a program, in the language you are most comfortable in, to retrieve the key used to encrypt multiple messages as well as decrypt said messages and an extra provided message.

3.2 Messages

Many Time Pad

Let us see what goes wrong when a stream cipher key is used more than once. Below are eleven hex-encoded ciphertexts that are the result of encrypting eleven plaintexts with a stream cipher, all with the same stream cipher key.

Hint: XOR the ciphertexts together, and consider what happens when a space is XORed with a character in [a-zA-Z].

ciphertext #1:

```
9d6e7a7d155295eef8512c087da56084f743aaa9985ee3848a768c3484d2e2ea6b3f4e5483f61
2d55987ff4782f360bd4809bab835fa1e65f8459c6814472508cc7f72241ee70c34fb9e7c8c4c
246132845d5d73d070de99a73efe1e9ee627ea8f4aaae147087cce6c8cd08813f81caf9d71204
86b16
```

ciphertext #2:

```
8968617d0f1b97e5ee51280f3aec72caf106bdb48d44a68291339e35db86b6f56f2140548bf25
3cd1593e8498baa34ba4f02fda070e25075b04387681b473610cc7f75614ae21a3efb8672d85d
3e2031cc5b5a23c17ad9cda334ef10
```

ciphertext #3:

```
8f6f3779154f99e8e014375c34a267c1e745bbad895fe391c526873383cfa1e86632575481e80
3c41c93f94d97a738f54d02a9ec66bb1360b44ed3210206254fcc627b240bfb1d38b899788a09
3d2f2a9b4b1327c17edf99b224fe1e9ee66effc649a3b5430c7c9a2a91d7c51bab14e9d57d395
66a5634c4340c858c5a93fff89e0394f3f49369ce9c6ee6f26b29a82e29fb9ca0a8657c48ad2c
60eaf70a6b44918a40630392171ca29b87cd9e9e037c6ad2bc4be08c61ba6223a8bac0a024741
b71b270a0579ab1d0308ce9b54b391a6a49ebacfc5eb5b57dd56caf0b1694544cfa6e
```

ciphertext #4:

```
9a697e6b415897fef902205c34a267d6fa42abbe985fe393972f963598c1b0fc7a3b4c17c2f10
1c51488f94199b667f54009b9eb7df40721ac4a963156473406cc62707406ea043cb586789c09
3f2f658d48433fc07ccacdaf23f54dd9a853e4df41ace6130e61986f8cd48156b11daad1692d4
33518349d2d0ec09a4ddafdf59e0882a4ffdd7cdd896bf7bb663ea83f37f59ca0e17a7a4aa96c
25fee50d2b0d93de5f6719d91700e19e97d186d7077c2985f856ef8035ab7c66bdbadfee2a664
e77bf66ac0788bac1748eacb45936196004a2b0e955b6a8798166e4
```

ciphertext #5:

```
8a68717e085ed8eae515651438a07fc9f448feb49358b19f812385249386b6f56f73461b8ce21
6dc0dc1e24ecfa361b74d0ebeeb7efe0921bb508a38024921118d7b757d44af1d31bed270995d
3e24288d4c5a30c8738b9bb23ef25d9caa27e4c908abfc550b678b2796d4891ab512a79d772c5
f2f5d3f872802cb895a93f7abd51993e5ee9376dbd072f0f27b35e43f2ffb85b7e4773242bb33
7caee40067079f934477149c520bfa9c81cf97d01c6125c6f352f88833af7466babc98e3247f4
b70ae7cee40c9b0cc2780bba25e325e694bacbfef59a5b27d8631
```

ciphertext #6:

```
9a6972380c5a91e5f80537193ca133c7e75faea9924bb191953e8f22d7c5adf067264b1d96f85
3c41892ad4480bd73f54902b1af35ef1860ac02972d05013543d93d306603fb4932be8b3d8f48
25613183571320c170d9cde638f41e80e173e3dc5caefb574d6fce688cc49113f515a6cf7f2c0
66e4c3385230885884ddcf3f898029fe1e8dd3fcc9f76f3a77d35fa2d75b281b7e56b7f45bf32
25e3ff003501d78d146e018e484ee18383d187ca0d6025d5f348ed9b61ae7f33acbfddf36b774
d60a86ca016d1f4c83b8dbdaf59795e624dbdbbf310a5b271813fa31400835f4eec250ddd43de
10a25e790c9f28e7ebd0c74fa11181fcf79cf68d5bd75b662bbacf38d31b39cbcdc71d213d611
812188ebe8855dcf857337ac75e2b36cceb0c4a729e1b7c72e78e0962c112351e62aa0f41456e
bd34667929cdb06c2ffd627d7b11da0b6dd57900d0cafd9651a32e8d4247a7ab3cce24d08150c
eee321b38dae50481acf00896ce1f8c7b7499dcf79008c9b5aee24b8de4ff1737116c
```

ciphertext #7:

```
876f377f04559df9ea1d695c2db971c8fc45feb69855e393972f963598c1b0fc7a3b5c5491f80
0d81c8cfe089aa071f54906afaf38ef1f2cab4d9f3e130636118369716107fc4938a8d269904c
7623249f514073c6798bcdae29bb5f9bef68f9c65ca7f81d4d5a866fdedc8a05ac53b9cf792d4
9625129852e17858f53d4f1aa9c1993e9bac770cb9162a3b46622a82e2ef09fbbeb2e7942a360
66fce91f330b978c557208805207f1cc9cd29392487064d6f95ba8862fea642eabf3c8f2227f5
e25bc74e35386a6d6748cafe75c320c7c04a7bfef57b4fa799b6baf1d06834901
```

ciphertext #8:

```
9c647079135f94eef80265133bec67ccf006b3bc8944a69d84228f2296cae2e962364a069ba11
1c91188e34ccfb27af5400bbaa467f20469b50ed33c1e436601897869240be30e36a99b699044
2561249e5d1327c170d8dce638f35f83a866f9ca08b8f05f0123856491c68b56b91dad9d6b2c4
a6315238b2316c88b51c7fbbcd50f9ee7fbc66ccad06febb77070e92c3eb292befb613250bf2c
69a3e40a3410959a14630e9d5219e780828c81ca1d766cc0f811
```

ciphertext #9:

```
9d64746a044fd8e0ee08651f2fb563d0fa41acbc8d44bad08833922998c2b1bd6f3e55188df85
3cd5992e44688bf71f54a02a4eb73f40221ba4d8720564328009e726d7003e00779ba9c79d84d
3322379548473ac67185
```

ciphertext #10:

```
8f2175740e5893abe818351438be33cde606adb2d04fa29c8933826195c3a1fc7f20405496e91
68c0a82e54d82b634b04f04afb265ef0321b74c9668144a2900872b72624aeb082dbad27c8c09
37613185555673dc6cc2d7a16cef5692a874eac24deffe56142e8164ded48415b053abd1732a4
d21
```

Target ciphertext (decrypt this one):

```
866e786a0042d4abf21e305c35ad65c1b542bbbe8f55b3848032c6359fc3e2e96b21421196a10
7c90195a30896bc61f54906abae35fd196fb1519b2d1206320b892b7c7719e60e37b697738c07
76292a9c5d132ac66a8bd1a728bb5882e629
```

For completeness, here is the python script used to generate the ciphertexts.

(it doesn't matter if you can't read this)

```
MSGS = ( ---  11 secret messages  --- )
def stringxor(msg1, msg2):     # xor two strings of different lengths
    if len(msg1) > len(msg2):
       return "".join([chr(ord(x) ^ ord(y)) for (x, y) in zip(msg1[:len(msg2)],
msg2)])
    else:
       return "".join([chr(ord(x) ^ ord(y)) for (x, y) in zip(msg1,
msg2[:len(msg1)])])

def random(size=16):
    return open("/dev/urandom").read(size)

def encrypt(key, msg):
    c = stringxor (key, msg)
    print
    print c.encode('hex')
    return c

def main():
    key = random(1024)
    ciphertexts = [encrypt(key, msg) for msg in MSGS]
```

4.0 Program Code Evaluation

4.1 Breakdown

50% Correct Target Decryption
30% Code Review
10% All Messages Decrypted
10% Full name and comments

Note!: Your program must be submitted via Canvas, and it must compile and run to receive credit. Programs that do not compile will receive an **automatic zero!**

4.2 Operational Code Required

Any compiler errors (e.g. gcc or javac is unable to compile the program code), as well as program run time failures or infinite loops will result **no credit** for affected parts of the Program Code for this Project. Please verify that your submission is operational prior to final submission.

4.3 Individual Work

The entire submission of Program Code and Project Report shall be your own individual original work. Any unauthorized use of shared, jointly written, or re-used code results in no credit for the entire Project.

5.0 Getting Started and Hints

5.1 OTP

A one time pad works by XORing a key with a given plaintext. The key is looped to fit the message if necessary. Should the key be used more than once the key can be retrieved and messages exposed.

Message 1 XOR key = Cipher 1
Message 2 XOR key = Cipher 2

(Cipher 1) XOR (Cipher 2) = (Message 1) XOR (Message 2)

5.2 Crib Dragging

Despite the insecurity that results from the XOR of two cipher texts. The resultant text is quite illegible. This is where crib dragging comes in, crib dragging is where you take frequently used words and xor them with the result of **(Cipher 1) XOR (Cipher 2)**. If the word is contained in one message then you will get the legible text of the other message. This portion will take some guessing as you may receive incomplete words or parts of sentences. Guess the rest of the word and then use that as the new crib. You will be uncovering the two, or more, of the messages involved simultaneously. When you have one completed message you can then use that to recover the original key.

5.3 ASCII, Hex, Binary

The project requires that at some point you XOR two given strings. These strings can be in ASCII, HEX, or Binary. The original strings are provided in HEX format. Feel free to convert, or not covert, between formats if it makes it easier.

5.4 Length and padding

An XOR is ,typically, done between two objects of the same length. You will notice that no two hex strings provided are the same length. Depending on the language used, there may ,or may not, be a third party library that will deal with this for you using padding. In either case, padding can either precede or come after that actual content. **Note:**

54686520736b792069732066616c6c696e67 **XOR** 4974206973206e6f740a0000000000000000

Does not equal

54686520736b792069732066616c6c696e67 **XOR** 00000000000000004974206973206e6f740a

6.0 Resources

6.1 Most frequent words in the English language

1	the
2	of
3	and
4	a
5	to
6	in
7	is
8	be
9	that
10	was
11	he
12	for
13	it
14	with
15	as
16	his
17	I
18	on
19	have
20	at

21	by
22	not
23	they
24	this
25	had
26	are
27	but
28	from
29	or
30	she

6.1 ASCII Table

http://www.asciitable.com/