Blake Johnson
9663980

Homework 1

Task 1:

a)This is a violation of <u>confidentiality</u> because the passwords that were supposed to be private but are now public. This is also a violation of <u>data integrity</u> because if the hacker wants to change something on your yahoo account they have that ability, however due to the volume of the hack, probably not all accounts would be changed. Having the password would also possibly allow the hacker to register a new computer with trusted certificates to other sites, or send emails from the account to others, violating <u>source integrity</u>. This would be similar to a recent ucsb umail attack where spam emails were being sent from hacked accounts to unhacked accounts serving the agenda of an unknown 3rd party.

b)This is a violation of <u>data integrity</u> because if there was information being written to a moving disk hard drive (which there probably was because this is a server room) then data would be lost. This would also be a violation of <u>availability</u>, because the information/services being hosted by the servers would not be able to send and receive with the power out.

c)This is a violation of <u>confidentiality</u> because the NSA could possibly view anything that is encrypted using AES. This is also a violation of <u>data integrity</u> because if the NSA can break AES, then they can change it and re-encrypt it to appear as if nothing had been changed, allowing their possibly malicious changes to go unnoticed.

d)<u>Availability</u> is the primary violation because John Smith cannot access the domain name that he represents. This is also potentially a violation of <u>source integrity</u> because if a 3rd party wants to interact with john smith they might assume they must visit JohnSmith.com, and interact with Anna, allowing her to assume John Smith's identity which can be used maliciously.

e) <u>Availability</u> is the primary violation, because the victim no longer has their data available to them because they cannot decrypt their own hard drive. Since the malicious user has the key to the hard drive, this is also a violation of <u>confidentiality and data integrity</u> because the malicious user can use the key to view/change data potentially.

f) This is a violation of <u>confidentiality</u> because the conversations and information sent and received are available to the NSA as well as the users they were intended for.

g)This is a violation of <u>confidentiality</u> because the foreign state actor can view who individuals voted for. This is also a violation of <u>data integrity</u> because the foreign state actor could possibly modify the votes that were sent in. This is a violation of <u>source integrity</u> because the people receiving the votes think they are receiving votes from citizens when they could actually be receiving votes from a foreign state actor.

Task 2:

a) Plaintext:
IFYOUTRYANDTAKEACATAPARTTOSEEHOWITWORKSTHEFIRSTTHINGYOUHAVEONYOU
RHANDSISANONWORKINGCATX

This plaintext was recovered using a brute force method of solving the Caesar cypher by trying all possible rotations to the alphabet until a readable decryption appears.

b) Plaintext:
output clean:
thechiefdifficultyalicefoundatfirstwasinmanagingherflamingoshesucceededingettingits
bodytuckedawaycomfortablyenoughunderherarmwithitslegshangingdownbutgenerall
yjustasshehadgotitsnecknicelystraightenedoutandwasgoingtogivethehedgehogablow
withitsheaditwouldtwistitselfroundandlookupinherfacewithsuchapuzzledexpressionth
atshecouldnothelpburstingoutlaughingandwhenshehadgotitsheaddownandwasgoingt
obeginagainitwasveryprovokingtofindthatthehedgehoghadunrolleditselfandwasinthea
ctofcrawlingawaybesidesallthistherewasgenerallyaridgeorfurrowinthewaywherevers h
ewantedtosendthehedgehogtoandasthedoubledupsoldierswerealwaysgettingupandw
alkingofftootherpartsofthegroundalicesooncametotheconclusionthatitwasaverydifficu
ltgameindeed

This plaintext was discovered using a mix of frequency analysis, reasoning about common words in English, and using this information to make initial mappings until enough of the words became clear and could be recognized. The most common letters in this string are GQAOLKSBPHTW. I have an interactive python script that prompts the most likely letter substitution but lets you map the letters on the go and reset if you make a mistake. Then once I was sure on a mapping, I hardcoded it in for future rounds so that I didn't have to remember it each time I reset or started the program. Initially I tried substituting different permutations of the most common English letters with the most commonly seen letters in the string. This did not illuminate enough of the string for me to make accurate guesses, so my next idea was to brute force. However, I quickly realized that the time complexity of that would be huge, so I settled on a mix of the two which was an interactive mix of frequency analysis and guess and check. I will attach my python code in a pdf

Task 3:

a) Yes, this is a valid block cipher because it is a one-to-one mapping of plaintext to ciphertext blocks per key value. The length of the ciphertext blocks is the same as the plaintext blocks, and for each key, there is only one ciphertext per plaintext, therefore ensuring this is a valid block cipher.
b) E' : $\{0,1\}^n$ x $\{0,1\}^n$ -> $\{0,1\}^n$ defines a mapping such that the key length is as long as the plaintext and ciphertext. The resulting substitution box will have 2^n rows and columns. Since E'(K,X) = X, the encryption is not reliant upon the key, making this block cipher an insecure substitution, despite technically being a valid block cipher because every number maps one-to-one to itself (e.g. 110 -> 110, 101->101, etc.)
c) E'': $\{0,1\}^n$ x $\{0,1\}^n$ -> $\{0,1\}^n$ defines a mapping such that the key length is as long as the plaintext and ciphertext, and the resulting substitution box will have 2^n rows and columns. Since E'(K,X) = Xnot xor K, is the encryption function, and since bitwise xor

of two strings produces a one-to-one mapping, each row will have a one-to-one mapping, and each row will be different because each key will produce different results when xor'd with the same plaintext.

d) E' is not a secure block cipher with any amount of bits, as the encryption function is a one to one mapping that doesn't modify the plaintext. However, with a 128-bit block length n, Xnot xor K can become any possibility with probability 1/(2^128), making this block cipher a secure block cipher.

Task 4:

a) With a pseudorandom iv, CBC mode, and using the pycrypto library, I encrypted the message X with itself as the key to be: ad77a337eadd6e56c1ee3477a15e2f38145867089548ef0247782eb8197046f4

b) I'm not sure if this is a trick question because due to the pseudo randomness of AES/CBC, the encryption of the message should be different each time it is encrypted or else it would violate the right definition of semantic security as stated by Goldwasser-Micali which means that nothing can be revealed about the message for it to be encrypted. So if a certain message would end in 00 each time it is encrypted, then that would reveal something about the message, because other messages would not be the same when encrypted. If the goal is purely to have the encryption end in 00, then the only thing you have to do is encrypt the same message multiple times until the randomness of the encryption encrypts the last byte to 00, but this is independent of the message.

c) There is no 16 byte key that always maps the last byte of an encrypted message to be 00, because this would break the encryption, because it would reveal information about the message, as stated by the right definition of semantic security by Goldwasser-Micali.

Task 5:

a) By saving the cipher text for 2^8 messages, the counter will get repeated because the counter is the date and it's only 8 bits. After this, we will know that one counter was used twice and we will know what message was sent originally, so we can xor the ciphertexts together to determine the second message, as the two counters that are the same will nullify each other when xor'd and we will be left with the plaintexts cor'd together. Since we know one of the plaintexts, we can determine the other to figure out what time they are meeting.

b) Mr. Cipher could protect himself by not using the same counter twice. To do this he needs to make his counter more bits so that he doesn't use all the possibilities. If he never uses the same counter twice he will be okay.

MY PYTHON FILE CONTAINING THE FUNCTIONS I USED FOR THIS HOMEWORK

```
import urllib.request
#from Crypto.Cipher import AES
#from Crypto import Random

URL1 = "http://www.cs.ucsb.edu/~tessaro/cs177/hw/cipher1.txt"
URL2 = "http://www.cs.ucsb.edu/~tessaro/cs177/hw/cipher2.txt"
URL3 = "http://www.cs.ucsb.edu/~tessaro/cs177/hw/cipher3.txt"
URLs = {URL1:"", URL2:"", URL3:""}

def shiftDecode(text):
  text = text.upper()
  alpha = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
  print(text)
  for i in range(26):
    newText = ""
    for letter in text:
      newLetter = alpha[(alpha.find(letter)+i)%26]
      newText += newLetter
    print(str(i) + " ")
    print(newText)

def monoSub(text):
  text = text.rstrip()
  text = text.upper()
  print(text)
  common = "ETAOINSHRDLU"
  alpha = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
  newMapping = ""
  frequency = {"A":0, "B":0,"C":0,"D":0,"E":0,
       "F":0,"G":0,"H":0,"I":0,"J":0,
       "K":0,"L":0,"M":0,"N":0,"O":0,
       "P":0,"Q":0,"R":0,"S":0,"T":0,
       "U":0,"V":0,"W":0,"X":0,"Y":0,
       "Z":0}
  for letter in text:
    frequency[letter] = frequency[letter] + 1


  max = -1
  maxLetter = ""
  descending = ""
  for i in frequency:
    max = -1
    for commonChar in frequency:
        if(frequency[commonChar] > max and commonChar not in descending):
            max = frequency[commonChar]
            maxLetter = commonChar
    descending+=maxLetter

min=0
minLetter = ""
for commonChar in frequency:
    if(len(newMapping)<12):
        print(newMapping)
        newMapping += commonChar
        if(frequency[commonChar] < min or len(newMapping)==1):
            min = frequency[commonChar]
            minLetter = commonChar
    elif(frequency[commonChar] > min):
        newMapping = newMapping.replace(minLetter, commonChar)
        min = frequency[commonChar];
        minLetter = commonChar
        for c in newMapping:
            if(frequency[c] < min):
                min = frequency[c]
                minLetter = c
    print(newMapping)
moreFrequent=""
lessFrequent=""

for i in range(6):
    maxMore = 0
    maxLetterMore = ""
    for c in newMapping:
        if(frequency[c] > maxMore and c not in moreFrequent):
            maxMore = frequency[c]
            maxLetterMore = c;
    moreFrequent += maxLetterMore
for i in range(6):
    maxLess = 0
    maxLetterLess = ""
    for c in newMapping:
        if(frequency[c] > maxLess and c not in moreFrequent and c not in lessFrequent):
            maxLess = frequency[c]
            maxLetterLess = c
    lessFrequent += maxLetterLess
    print("moreFrequent "+ moreFrequent)
    print("lessFrequent "+ lessFrequent)

    for letter in moreFrequent:
        print(letter + " " + str(frequency[letter]))

    for letter in lessFrequent:
        print(letter + " " + str(frequency[letter]))


    moreCommon = ["etaoin","etaoni","etaion","etaino","etanoi","etanio",
                  "etoian","etoina","etoain","etoani","etonia","etonai"]

    lessCommon = ["shrdlu","shrdul","shrldu","shrlud","srhdlu","srhdul",
                  "srhldu","srhlud","hsrdlu","hsrdul","hsrldu","hsrlud"]


    ##Interactive Method

    output = text
    Frequent = descending
    i=0
    mapping = {"e":"G","t":"Q","a":"","o":"","i":"","n":"","h":"S","s":"","r":"","d":"","l":"","u":""}
    print("Frequent: "+Frequent)
    print(mapping)
    option = input("Enter a letter to replace the letter "+ Frequent[i] +" next or enter 'quit' to
quit")
    while(option != "quit"):
        output = output.replace("Q","t")
        output = output.replace("G","e")
        output = output.replace("S","h")
        output = output.replace("H", "g")
        output = output.replace("W", "r")
        output = output.replace("M", "b")
        output = output.replace("Z", "u")
        if(option == "reset"):
            output = text

            i=0
            for entry in mapping:
                mapping[entry] = ""
        else:
            print("Replacing "+Frequent[i]+" with "+option)
            mapping[option] = Frequent[i]
            output = output.replace(Frequent[i], option)
            outputclean = output
            for c in outputclean:
                if(c.isupper()):
                    outputclean = outputclean.replace("Q","t")
                    outputclean = outputclean.replace("G","e")
                    outputclean = outputclean.replace("S","h")
                    outputclean = outputclean.replace("H", "g")
                    outputclean = outputclean.replace("M", "b")
                    outputclean = outputclean.replace(c, "*")


            print("output: "+output)
            print("output clean: " + outputclean)
            i += 1
            print(mapping)
        option = input("Enter a letter to replace the letter "+ Frequent[i] +" next, enter 'quit' to
quit, or enter 'reset' to start over")




    ##ORIGINAL METHOD

    #outputBoth = ""
    #for j in range(12):
    #    output = text
    #    #output = output.replace("S", "h")
    #    for i in range(6):
    #        output = output.replace(moreFrequent[i], moreCommon[j][i])
    #        #print("replacing "+moreFrequent[i]+" with "+moreCommon[j][i])
    #    for k in range(12):
    #        outputBoth = output
    #        for l in range(6):
    #            outputBoth = outputBoth.replace(lessFrequent[l], lessCommon[k][l])
    #            #print("replacing "+lessFrequent[l]+" with "+lessCommon[k][l])
    #        if("the" in outputBoth and "and" in outputBoth and "in" in outputBoth and "that" in
outputBoth):
    #            print(moreFrequent+lessFrequent)
    #            print(moreCommon[j]+lessCommon[k])
    #            print(outputBoth)
    #            print("\n")
```

```
##BRUTE FORCE
    # print("BRUTE FORCE")
    # max = -1
    # maxLetter = ""
    # descending = ""
    # print("Frequency")
    # print(frequency)
    # for i in frequency:
        # max = -1
        # for commonChar in frequency:
            # if(frequency[commonChar] > max and commonChar not in descending):
                # max = frequency[commonChar]
                # maxLetter = commonChar
        # descending+=maxLetter


    # print(descending)
    # for c in descending:
        # print(c + " " + str(frequency[c]))

    # output = text
    # frequentLetters = "etaoinsrhldcumfpgwybvkxjqz"
    # mapping =
{"e":"","t":"","a":"","o":"","i":"","n":"","s":"","r":"","h":"","l":"","d":"","c":"","u":"","m":"","f":"",
"p":"","g":"","w":"","y":"","b":"","v":"","k":"","x":"","j":"","q":"","z":""}





# def mapFunc(text, commonEng, commonText, index):
        # output = text
        # output = output.replace(commonText[index], common)



def setupURLs(URLs):
    for entry in URLs:
        data = urllib.request.urlopen(entry)
        cipherText = data.read().decode("utf-8")
        URLs[entry] = cipherText

setupURLs(URLs)

# for url in URLs:
    # print(URLs[url])

#def task4a():
#    key = bytes.fromhex("10042018000000000000000000000000")
#    iv = Random.new().read(16)
#    AEScipher =  AES.new(key, AES.MODE_CBC, iv)
#    message = iv + AEScipher.encrypt(key)
#    output = message.hex()
#    print(output)


#Task 2a
#shiftDecode(URLs[URL1])

#Task 2b
monoSub(URLs[URL2])

#Task 4
#task4a()
```