CSEP590 : Applied Cryptography: Homework 1

Karuna Sagar Krishna

April 15, 2023

Task 1a

The plain text is -

```
icheckeditverythoroughlysaidthecomputerandthat
quitedefinitelyistheanswerithinktheproblemtobequitehonest
withyouisthatyouveneveractuallyknownwhatthequestionis
```

At first it looks like jumbled text, but that is simply because it is missing spaces and punctuations. Adding spaces to make it easier to read the plain text -

```
i checked it very thoroughly said the computer and that
quite definitely is the answer i think the problem to be
quite honest with you is that youve never actually known
what the question is
```

As described in the problem statement, there are only 26 possibilities for the secret key. We can enumerate all keys and run the decryption algorithm. The shift cipher encryption algorithm shifts each character in plain text to the right by k steps in alphabetical order. So the decryption algorithm should shift the cipher text character to the left by k steps. Since we running the decryption algorithm for all possible keys, it produces the same set of output plain texts if we instead ran the encryption algorithm. This is because decryption with key $= k$ is the same as encryption with key $= 26 - k$.

The following code runs the encryption algorithm for all possible keys and prints the possible plain texts. We are told that the plain text is readable English sentence as per EdStem discussion. So, we can filter out the plain texts manually to find the right plain text. There was only one readable English sentence.

```python
cipher_text = "" # paste cipher text here
for k in range(26):
    plain_text = ""
    for c in cipher_text:
        plain_text += chr(
            ((ord(c) - ord('a') + k) % 26) + ord('a'))
    print("{} : {}".format(k, plain_text))

```

Task 1b

The plain text is -

```
    itisanimportantandpopularfactthatthingsarenotalwayswhat
theyseemforinstanceontheplanetearthmanhadalwaysassumed
thathewasmoreintelligentthandolphinsbecausehehadachievedso
muchthewheelnewyorkwarsandsoonwhilstallthedolphinshadeverdone
wasmuckaboutinthewaterhavingagoodtimebutconverselythedolphins
hadalwaysbelievedthattheywerefarmoreintelligentthanmanfor
preciselythesamereasonscuriouslyenoughthedolphinshadlongknown
oftheimpendingdestructionoftheplanetearthandhadmademany
attemptstoalertmankindofthedangerbutmostoftheircommunications
weremisinterpretedasamusingattemptstopunchfootballsorwhistle
fortidbitssotheyeventuallygaveupandlefttheearthbytheirownmeans
shortlybeforethevogonsarrivedthelasteverdolphinmessagewas
misinterpretedasasurprisinglysophisticatedattempttodoadouble
backwardssomersaultthroughahoopwhilstwhistlingthestarspangled
bannerbutinfactthemessagewasthissolongandthanksforallthefish
infacttherewasonlyonespeciesontheplanetmoreintelligentthan
dolphinsandtheyspentalotoftheirtimeinbehaviouralresearch
laboratoriesrunningroundinsidewheelsandconductingfrighteningly
elegantandsubtleexperimentsonmanthefactthatonceagainman
completelymisinterpretedthisrelationshipwasentirelyaccordingto
thesecreaturesplans
```

Adding spaces for better reading -

```
it is an important and popular fact that things are not
always what they seem for instance on the planet earth
man had always assumed that he was more intelligent than
dolphins because he had achieved so much the wheel
newyork wars and so on whilst all the dolphins had ever
done was muck about in the water having a good time but
conversely the dolphins had always believed that they
were far more intelligent than man for precisely the
same reasons curiously enough the dolphins had long
known of the impending destruction of the planet earth
and had made many attempts to alert mankind of the danger
but most of their communications were misinterpreted as
```

```
amusing attempts to punch footballs or whistle for
tidbits so they eventually gave up and left the earth
by their own means shortly before the vogons arrived the
last ever dolphin message was misinterpreted as a
surprisingly sophisticated attempt to do a double
backwards somersault through a hoop whilst whistling
the star spangled banner but infact the message was this
so long and thanks for all the fish infact there was only
one species on the planet more intelligent than dolphins
and they spent a lot of their time in behavioural research
laboratories running round inside wheels and conducting
frighteningly elegant and subtle experiments on man the
fact that once again man completely misinterpreted this
relationship was entirely according to these creatures
plans
```

As discussed in class, we can use frequency analysis to break mono-alphabetic substitution ciphers. So firstly, we need the frequency of English language. I used the frequency table from the Internet. Then I computed the frequency tables for the given cipher text using the code below.

```python
cipher_text = ""

freq = {}
for c in cipher_text:
    if c not in freq:
        freq[c] = 0
    freq[c] += 1

difreq = {}
for i in range(0, len(cipher_text)):
    if i + 1 >= len(cipher_text):
        break
    di = cipher_text[i] + cipher_text[i + 1]
    if di not in difreq:
        difreq[di] = 0
    difreq[di] += 1

trifreq = {}
for i in range(0, len(cipher_text)):
    if i + 2 >= len(cipher_text):
        break
    tri = cipher_text[i] + cipher_text[i + 1] + cipher_text[i + 2]
    if tri not in trifreq:
        trifreq[tri] = 0
    trifreq[tri] += 1

quadfreq = {}
for i in range(0, len(cipher_text)):
    if i + 3 >= len(cipher_text):
        break
    quad = cipher_text[i] + cipher_text[i + 1] +
            cipher_text[i + 2] + cipher_text[i + 3]
    if quad not in quadfreq:
        quadfreq[quad] = 0
    quadfreq[quad] += 1

print("Frequency: {}".format(
    sorted(freq.items(), key=lambda x: x[1], reverse=True)))
print("Di-frequency: {}".format(
    sorted(difreq.items(), key=lambda x: x[1], reverse=True)))
print("Tri-frequency: {}".format(
    sorted(trifreq.items(), key=lambda x: x[1], reverse=True)))
print("Quad-frequency: {}".format(
    sorted(quadfreq.items(), key=lambda x: x[1], reverse=True)))

```

Next, I wrote the below code to decrypt the cipher text assuming we have the exact substitution determined. Now, the hard part was to fill up the subsitution dictionary. As discussed in EdStem, the plain text is a readable English text. Note, since the cipher text contained only upper case letters I chose to map them to lower case letters to help distinguish cipher and plain text. With this assumption, I started the effort to figure out the substitutions by mapping the most frequent letters in English to the most frequent letters in the cipher text in order. This was done blindly and in retrospect this worked out well, but this mapping could have gone wrong. I ran the below code to see if I can manually infer any other clues. Next I used the bigram/digram frequency table since I already had a mapping for cipher text 'R', I guessed cipher text 'N' would map to 'h' since 'th' is the most frequent bigram/digram. Similarly, with the bigram/digram frequency I mapped 'E' to 'n' since 'an' is a frequent bigram. After each update to substitution dictionary I ran the decryption and stared at the partial plain text to find any new clues. Then I noticed the trigram 'QEW' was frequent in cipher text and this meant I should map 'W' to 'd' since 'and' is a frequent trigram in English. Next, I noticed that 'GE' is cipher text frequency table and comparing it with English frequency table we should map 'G' to 'i'. At this point the partial plain text was quite filled out (lesser ? characters). Next clue came from trigram 'GEI' which indicated that 'I' should map to 'g'. At this point I couldn't find any further clues from the frequency table. Reading the first part of the partial plain text (*iti?ani????tantand?????a??a?*), hinted that maybe 'Z' should map to 's'. With this guess, the partial plain text started becoming more readable. The frequency table was getting harder to use but since the parital plain text was more readable, I started to guess the remaining letters and which each guess it became easier to guess the next mapping. Eventually, there were no more '?' characters in the plain text indicating that we have found the substitution needed to decrypt the given cipher text. Using vscode to match the columns of the partial plain text and cipher text helped with the manual guess effort. Finally, I verified that the plain text was readable English and I assumed that there is only one substitution that leads to a readable English.

```
subst = {
    'T': 'e', # 'e' is the topmost frequent letter in English
    'R': 't', # 't' is the next frequent letter in English
    'Q': 'a', # 'a' is the next frequent letter in English
    'N': 'h', # guess based on bigram/digram frequency
    'E': 'n', # guess based on bigram/digram frequency
    'W': 'd', # guess based on trigram frequency
    'G': 'i', # guess based on bigram/digram frequency
    'I': 'g', # guess based on trigram frequency
    'Z': 's', # guess based on my reading of the partial plain text
    'P': 'r', # guess based on my reading of the partial plain text
    # all below mappings were guessed manually reading the partial
    plain text
    'J': 'o',
    'O': 'c',
    'K': 'm',
    'X': 'p',
    'D': 'l',
    'L': 'u',
    'H': 'f',
    'V': 'w',
    'Y': 'y',
    'U': 'b',
    'F': 'v',
    'S': 'k',
    'B': 'x'
    }
plain_text = ""
for c in cipher_text:
    if c in subst:
        plain_text += subst[c]
    else:
        plain_text += '?'

print(plain_text)

```

Task 2a

```
24 f3 dc 26 07 11 10 ad 52 58 a4 55 67 14 d0 1d
```

Using the code snippet from Lecture 1, we run the AES in ECB mode. $AES(X, X)$ implies that both the key and message are exactly the same 16 bytes. To compute this we simply call the code snippet appropriately as shown below.

```
X = bytes.fromhex('10 04 20 18 00 00 00 00 00 00 00 00 00 00 00 00'
    .replace(' ', ''))
key = X
pt = X

import os
from cryptography.hazmat.primitives.ciphers import Cipher,
    algorithms, modes
from cryptography.hazmat.backends import default_backend
```

```

backend = default_backend()
cipher = Cipher(algorithms.AES(key), modes.ECB(), backend=backend)
encryptor = cipher.encryptor()
ct = encryptor.update(pt) + encryptor.finalize()
print(ct.hex())

```

Task 2b

```
fd e4 d4 2d 80 2d 9e 09 18 fd 5f ae 0c 6f a2 9c
```

Given key as X, we want to find plain text M such that the last 8 bytes of cipher text are 00. Since there is no restriction on the first 8 bytes of the cipher text, lets assume cipher text is 0^{128}. We can simply decrypt this cipher text to find the plain text M hence answering the question. To decrypt, we use the below code. We can verify this by encrypting M using the same key.

```
X = bytes.fromhex('10 04 20 18 00 00 00 00 00 00 00 00 00 00 00 00'
    .replace(' ', ''))
key = X
ct = bytes.fromhex('00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
    '.replace(' ', ''))

import os
from cryptography.hazmat.primitives.ciphers import Cipher,
    algorithms, modes
from cryptography.hazmat.backends import default_backend

backend = default_backend()
cipher = Cipher(algorithms.AES(key), modes.ECB(), backend=backend)
decryptor = cipher.decryptor()
pt = decryptor.update(ct) + decryptor.finalize()
print(pt.hex())

```

Task 2c

```
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 4b
```

The questions asks to find the key given a plain text and cipher text. Note, we are told that last byte of cipher text has to be 00 and since there is no restriction on other bytes we can assume it. Hence the cipher text is known.

With this information, it seems impossible to find the key, because if there some way to do that people would have broken AES. The only thing that remains is a brute force attack and hopefully we find the key in reasonable amount of time. Fortunately this turned out to be true. The below python code starts with key 0^{128}, checks to see if AES produces a cipher text with last byte as 00. If not, it increments the key and tries again. Fortunately this ran for few seconds until it stopped and found the key satisfying the question. To verify we can run the AES encryption using this key and X to see that the last byte of cipher text is 00

```python
k = bytes.fromhex('00000000000000000000000000000000')
c = aes(k, X)
while not c.endswith(b'\x00'):
    k = (int(k.hex(), 16) + 1).to_bytes(16, 'big')
    c = aes(k, X)
print(k.hex())

```

Task 3

System $S_0.Eval()$ has 4 possible outputs - $\{00, 01, 10, 11\}$. This is because b_1 can be either 0 or 1 and similarly b_2 can be either 0 or 1. The random selection of b_1 and b_2 is independent of each other since the choice of b_1 does not affect the choice of b_2. Hence the probability of $S_0.Eval()$ returning $b_1||b_2$ is $Pr(b_1||b_2) = Pr(b_1) * Pr(b_2)$. So the probability of each output is $\frac{1}{4}$.

System $S_1.Eval()$ has 3 possible outputs - $\{00, 01, 10\}$. As we see from the $Init()$, the value of b_1 is chosen randomly so there is $\frac{1}{2}$ probability for $b_1 = 0$ and $b_1 = 1$. If b_1 is 0, then value of b_2 is chosen randomly similar to S_0, but when the value of b_1 is 1, b_2 is fixed to 0. Hence the probabilities of output 00 and 01 is $\frac{1}{2} * \frac{1}{2} = \frac{1}{4}$ and the probability of output 10 depends only on the probability of choosing b_1 which is $\frac{1}{2}$.

Task 3a

Attacker A_1 only looks at the first bit of the output of $Eval()$. So,

$$\begin{aligned}
Pr(A_1(S_0) = 1) &= Pr(S_0.Eval() = 10 \vee S_0.Eval() = 11) \\
&\qquad \triangleright \textit{ we are dealing with disjoint events} \\
&= Pr(S_0.Eval() = 10) + Pr(S_0.Eval() = 11) \\
&= \frac{1}{4} + \frac{1}{4} \\
&= \frac{1}{2}
\end{aligned}$$

and

$$\begin{aligned}
Pr(A_1(S_1) = 1) &= Pr(S_1.Eval() = 10) \\
&= \frac{1}{2}
\end{aligned}$$

Hence, the distinguishing advantage is 0 i.e. attacker A_1 is too naive and it cannot distinguish between S_0 and S_1.

$$\begin{aligned} Adv^{dist}_{S_0,S_1}(A_1) &= |Pr(A_1(S_0) = 1) - Pr(A_1(S_1) = 1)| \\ &= |\frac{1}{2} - \frac{1}{2}| \\ &= 0 \end{aligned}$$

Task 3b

Attacker A_2 is more complex than A_1 since it does an XOR operation of the two bits returned by $Eval()$. So,

$$\begin{aligned} Pr(A_2(S_0) = 1) &= Pr(S_0.Eval() = 01 \lor S_0.Eval() = 10) \\ &\triangleright \textit{ we are dealing with disjoint events} \\ &= Pr(S_0.Eval() = 01) + Pr(S_0.Eval() = 11) \\ &= \frac{1}{4} + \frac{1}{4} \\ &= \frac{1}{2} \end{aligned}$$

and

$$\begin{aligned} Pr(A_2(S_1) = 1) &= Pr(S_1.Eval() = 01 \lor S_1.Eval() = 10) \\ &\triangleright \textit{ we are dealing with disjoint events} \\ &= Pr(S_1.Eval() = 01) + Pr(S_1.Eval() = 10) \\ &= \frac{1}{4} + \frac{1}{2} \\ &= \frac{3}{4} \end{aligned}$$

Hence, the distinguishing advantage is $\frac{1}{4}$ i.e. attacker A_2 is able to distinguish between S_0 and S_1 with 25% probability.

$$\begin{aligned} Adv^{dist}_{S_0,S_1}(A_2) &= |Pr(A_2(S_0) = 1) - Pr(A_2(S_1) = 1)| \\ &= |\frac{1}{2} - \frac{3}{4}| \\ &= \frac{1}{4} \end{aligned}$$

Task 4a

As hinted in the question, we use the generic attack discussed in class as the concrete attack for this task. The generic attack is brute force attack that searches for a key that is consistency with the outputs as shown below.

```
**procedure** $A_3(S)$
    $y_0 = S.Eval(0^n)$
    $y_1 = S.Eval(1^n)$
    **if** $\exists K : E(K, 0^n) = y_0 \wedge E(K, 1^n) = y_1$ **then**
        **return** 1
    **else**
        **return** 0
```

The probability for this attacker to recognize the system using cipher E from a random function is 1. Because the attacker enumerates through all possible keys, so we will eventually find some key that is consistent with the outputs y_0 and y_1.

$$Pr(A_3(S_0) = 1) = 1$$

Given a key, the probability that $E(K, 0^n) = y_0$ is 2^{-n} since there only one way this can happen out of 2^n possibilities. Then the probability that $E(K, 1^n) = y_1$ is $\frac{1}{2^n-1}$ because block cipher E would give us a random permutation and since we already sampled $E(K, 0^n)$ we only have $2^n - 1$ possibilities. Hence for a given key, the attacker has probability of $\frac{1}{2^n} * \frac{1}{2^n-1}$. Considering all possible keys i.e. $2^k = 2^n$ keys, we get a probability of $\frac{1}{2^n} * \frac{1}{2^n-1} * 2^n \leq \frac{1}{2^n}$. So, the probability for this attacker to recognize a truly random function is less than 2^{-n}.

$$Pr(A_3(S_1) = 1) \leq 2^{-n}$$

Hence, with $t = O(2^n)$ and $q = 2$, the attacker has very good chance of distinguishing the system i.e. distinguishing advantage for this attacker is $Adv_E^{prf}(A) = 1 - 2^{-n}$

$$\begin{aligned} Adv_E^{prf}(t, q) &= \forall A, max[Adv_E^{prf}(A)] \\ &\geq 1 - 2^{-n} \\ &\geq \frac{1}{2} \end{aligned}$$

This generic attack applies to AES as well, but note that t is very large since the attacker has to enumerate all possible keys. So a powerful attacker can distinguish AES from random function. AES encrypts 128 bits at a time, so $n = 128$ and $t = 2^{128}$ which is very large amount of time (about a billion years). Hence, AES is secure for a computationally bounded attacker.

Task 4b

Consider the following concrete attack on E which satisfies $E(K, X \oplus X') = E(K, X) \oplus E(K, X')$

```
procedure $A_4(S)$
    $p_1 = \{0,1\}^n$
    $p_2 = \{0,1\}^n$
    $p = p_1 \oplus p_2$
    $c = S.Eval(p)$
    if $c = E(p_1) \oplus E(p_2)$ then
        return 1
    else
        return 0
```

The probability for this attacker to recognize S_0 (representing the block cipher E) is 1 because the attacker leverages the knowledge that the E is linear.

$$Pr(A_4(S_0) = 1) = 1$$

The probability for this attacker to recognize S_1 (representing a truly random function) is 2^{-n} because there is only way for $S.Eval$ to return an output where $S.Eval(p_1 \oplus p_2) = E(p_1) \oplus E(p_2)$ holds.

$$Pr(A_4(S_1) = 1) = 2^{-n}$$

With $t = O(n)$ and $q = 1$, the distinguishing advantage of this attacker is high. So a computationally bounded attacker can clearly know when the system is using a block cipher E which is linear.

$$\begin{aligned} Adv_E^{prf}(t,q) &= \forall A, max[Adv_E^{prf}(A)] \\ &\geq 1 - 2^{-n} \\ &\geq \frac{1}{2} \end{aligned}$$

Task 4c

The block cipher given in this sub-task has similarities to EBC mode i.e. it breaks up the input bit string and then applies two different functions and concatenates the result as output. Consider the following concrete attack:

```
procedure $A_5(S)$
    $p_1 = \{0,1\}^{64}$
    $p_2 = \{0,1\}^{64}$
    $p_3 = \{0,1\}^{64}$
    $p = p_1||p_2$
    $p' = p_1||p_3$
    $c = S.Eval(p)$
    $c' = S.Eval(p')$
    $c_1||c_2 = c$
    $c'_1||c'_2 = c'$
    if $c_1 = c'_1$ then
        return 1
    else
        return 0
```

The probability for this attacker to recognize S_0 is 1 because the attacker leverages the knowledge of block cipher algorithm.

$$Pr(A_5(S_0) = 1) = 1$$

The probability for this attacker to recognize S_1 is 2^{-64}. Note, we need to match the first 64 bits of $S.Eval(p)$ and $S.Eval(p')$ i.e. the last 64 bits can be anything and we have 2^{64} options. Since S_1 is a random function, the output of $S.Eval(p')$ is not restricted since we already sample $S.Eval(p)$ i.e. we have $2^n = 2^{128}$ choices for $S.Eval(p')$. Hence we get

$$\begin{aligned} Pr(A_5(S_1) = 1) &= 2^{64}/2^{128} \\ &= 2^{-64} \end{aligned}$$

With $t = O(n)$ and $q = 2$, the distinguishing advantage of this attacker is high. So a computationally bounded attacker can clearly know when the system is using a block cipher E which is linear.

$$\begin{aligned} Adv_E^{prf}(t,q) &= \forall A, max[Adv_E^{prf}(A)] \\ &\geq 1 - 2^{-64} \\ &\geq \frac{1}{2} \end{aligned}$$

Task 4d

No, if we remove any of the components of AES, we don't get a good PRF security. Lets consider the following cases to see this.

Remove SubBytes

So each AES round only performs ShiftRows and MixColumns operation. As noted in class both of these are linear operations i.e. the output of AES can be expressed as linear equation involving the input and key. If we can find sufficient number of input and output i.e. plain text and cipher text, then we get a system of linear equations which can be solved to find the key.

Remove ShiftRows

As discussed in class, the 16 byte plain text is represented in matrix form. This state matrix goes through various rounds. SubByte transforms each byte of the state using S-box. Then MixColumns performs a linear transformation of each column independently. Since we are removing ShiftRows, each column doesn't influence the other despite xor-ing the round keys. This implies that we can attack each column independently. Each column has 4 bytes (32 bits) and we can perform a brute force attack to explore all 2^{32} possible permutations to find the plain text. Essentially, this modified cipher breaks up the 16 byte plain text, permutes each 4 byte block and concatenates them as output. Hence the attack would be similar to the penguin attack on EBC.

Remove MixColumns

Similar to removing ShiftRows, when MixColumns operation is removed, the values in each row don't influence the other rows. This holds though we xor the round keys. Since the 16 byte plain text was written in column major format to form the state matrix, each byte ends up being independent of the others. So we can attack each byte independently by exploring all possible 2^8 permutations for a given byte. Essentially, this modified cipher breaks up the 16 byte plain text, permutes each byte and concatenates them as output. Hence the attack would be similar to the penguin attack on EBC.