Assignment3

Wonchan Kim 301449586

Q1

a.
   Superfish installs its own root certificate into the local trusted certificates store. Hence, the superfish could act as man in the middle between the user's browser and any HTTPS website. As given in the hint, if the user is talking to the website using the website's encryption key, Superfish can't modify the contents because Superfish doesn't have website's decryption key. So instead, browser connects to Superfish instead of directly to the website. Superfish then connects separately to the real website using a legitimate HTTPS connection and receives the genuine encrypted content (intercept).Superfish then decrypts the content using the session established with the real website, then reads and modifies the webpage. It then encrypts the modified data using a certificate that is generates for the website, signed by its own trusted root certificate. As the browser already trusts the Superfish's root certificate, it accepts the fake certificate and allows the modification.
b. Superfish used the same signing key pair on every Lenovo laptop. If an attacker could extract the private key which was the same for everyone, then attacker can use the superfish root private key to generate a fake certificate for any domain. Since every Lenovo laptop with Superfish trust the root certificate, the fake certificate will be accepted as valid. Then the attacker can now conduct Man-in-the-middle attack, presenting the forged certificate and decrypting user's HTTPS traffic.
c. Careful users could detect in several ways; when clicking the padlock icon on the browser, the user can check the details of certificate. The user would then notice the certificate for the specific website was issued by the Superfish. Also user would be able to check installed certificates in the browser's certificate manager. The user would be able to find the root certificate named Superfish.

> Q2

Used Index of Coincidence method for this question. It measures the probability that two randomly selected letters from a text are the same. In normal English, the IC is around 0.066, while in completely random text is about 0.038.

```
> with(StringTools):

  yRaw :=
  "TQYBRXGLBFGQQFBUNWTPKGSTLTOVBNSLQMUXCTGWAYFTHARAPMDSMITGKDNLUATXZ
  HAPDGTSMXNDGYDWXCOGTRJVIUQGCJODFGWSTFPFJXLFMXOUBRHHBYQGGTPTHGKHYBH
  ARMHTGPXYADSMWQTXHXHCXAFZYATYCMHHTAOQGSJGIBTXRJGIMUBCJYGAOKDKXGQPV
  DXHGQZMDWGPXNBMPLBMVADRTIUETKHHCFGGSXADGNWBTFTNGYNWXRAPMDSMITCMCTX
  HZQMCJTAUPITWXBMVADRTIUELHSVAGFBMLIGMEMHHTAGUXRNGPBREHJWUUGECXLJOJ
  TRXVXQPVDFGSQZMDSWXZIMNHNAFWKZQTHEQVHFMXAPLZSWITGAHXMDDAHEXRBNQERF
  LHAEBZYXSIKMGSNBNGKRRTITGFZYBRMNVNAXGMIXRMHJXFHOJGLUVARNFEXGFZYATY
  CMHHTABTHOJKIUGLNKMWQPNLGXGMPWKNLIEQFDXBBBNXZXLDOKTSJWEDQIDWMXQUPG
  JMWQTMGJRPDGIQNFTATGNYTQGPWZSMTFEMGJGPDVBBQXHEJHTQWBAXXNSMDPKLBZLH
  PGXOJKBMVADRTIUELSMTIUUTRXHRUCMDIPXFJMGJGJYDXQXLJOJTRUKDBGKSNXHMTB
  RNGVRTHLGXXZITLJFQQTHEXITOKYHHVAMULDXHUBTBLJLDDRKNUXGFKXRYAPFUMDRY
  GAOKDQTIUQGRMBEEYBSMIPDVBBZEPDIXNRXIDKVENZJDGLZSRGQNXUFGIYCMGJFPFK
  VZQAXEVHQDHUFJXMZFQQTTRNMGQNTSJLIAVADXXEDQIDWMXQUBRFEHAYHQYARAXXQN
  GVFJXZIOXEGYNWTHEGLRNGVYCMGJFPFKVZQVDZVXMYLEQEBENVPXNRCNLIUPZTNLWQ
  UUDYPTQPVNSMTZVMGFMXEUBLUETATUZXBRRQKLZEPUETMIHURVHONVGQHXQJGRUPZS
  MXDZNBMJXCOAVKTITPKTNKBCFGZDWLTCWXMHXHAGBRNLAMTZDQRPPOBRXBQXGTMIXC
  OQNQFZTPYBSMBCXKFHYLRDKMDWBPSGGDWTAXAYNQEDIUFDWBIEQYTSBFGGGDXLDRVA
  DSNBNGKRNGFGGLSNHCIKMGNGITGLDVNTZEXRFGSMPRNYATDTXKJOPZVTRXHRUCMDIL
  TCWXMHXHMULDXLXZIWDJITDEHMSXRFKHMXHUUPMDLXGEVHNYATDOTSMXBMVBBFEDNL
  XBYLDRKGSJKTEVBRFEHATXKJOPZVTMITCUOINWMPZVIZWMDR
  EHUJKPSGHERTITGFZYBRMNIQTITDVBDXTHOTBAJW
  IAVADSNBNGKRYAXEINHIXAUPXZQLDQOIGFLXLGLOJKLUMBOJWXMRHKNVNFJTSTKXSK
  GZQKTEGTQHAXZEETXBKQQYRDGITGLHXHUBWUKNLWQFBMKHGYCMHTGXEPHSUXGYKMSJ
  W":

  # Fileter only Characters
  yNew := cat(seq(if (c >= "A" and c <= "Z") then c else "" end if, c
  in Explode(yRaw))):

  n := length(yNew):
  printf("Ciphertext length (n) after filtering: %d\n", n); # 1494
```

```
Ciphertext length (n) after filtering: 1474
```

Using Index of Coincidence to find block size m

For a candidate m we split the ciphertext into m subsequences by taking every m-th letter, so each subsequence contains letters encrypted with the same key letter. If m equals the true key length, each subsequence behaves like a monoalphabetic (Caesar) encryption of English and therefore has IC close to English.

```
> Sigma := "ABCDEFGHIJKLMNOPQRSTUVWXYZ":
  for i from 0 to 25 do
      char[i] := Sigma[i+1]:
      code[char[i]] := i:
  od:

  # --- 1. Index of Coincidence (IC) ---
  M_check := 15:
  Ic := Matrix(M_check, M_check):

  print("Calculating Index of Coincidence (IC) for m=1 to 15...");
  for m to M_check do
      for i to m do
          yy_seq := seq(yNew[j], j=i..n, m):
          yy := cat(yy_seq):
          nn := nops([yy_seq]):

          if nn < 2 then
              Ic[m,i] := 0.0:
          else
              f_count := Array(0..25, fill=0):
              for j to nn do
                  f_count[code[yy[j]]] := f_count[code[yy[j]]] + 1:
              od:
              Ic[m,i] := evalf(add(binomial(f_count[k],2), k=0..25) /
  binomial(nn,2), 3):
          end if:
      od:
  od:

  print("--- IC Matrix (Rows = key length m) ---");
  print(Ic)
```

```
"Calculating Index of Coincidence (IC) for m=1 to 15..."
"--- IC Matrix (Rows = key length m) ---"
```

$$\begin{array}{c|cccccccccc c}
 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & \\
1 & 0.0428 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \cdots \\
2 & 0.0419 & 0.0438 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \cdots \\
3 & 0.0439 & 0.0425 & 0.0442 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \cdots \\
4 & 0.0422 & 0.0445 & 0.0417 & 0.0434 & 0 & 0 & 0 & 0 & 0 & 0 & \cdots \\
5 & 0.0441 & 0.0434 & 0.0385 & 0.0442 & 0.0443 & 0 & 0 & 0 & 0 & 0 & \cdots \\
6 & 0.0444 & 0.0447 & 0.0441 & 0.0420 & 0.0397 & 0.0466 & 0 & 0 & 0 & 0 & \cdots \\
7 & 0.0731 & 0.0797 & 0.0730 & 0.0683 & 0.0640 & 0.0595 & 0.0717 & 0 & 0 & 0 & \cdots \\
8 & 0.0456 & 0.0432 & 0.0425 & 0.0441 & 0.0410 & 0.0462 & 0.0391 & 0.0417 & 0 & 0 & \cdots \\
9 & 0.0522 & 0.0438 & 0.0439 & 0.0382 & 0.0403 & 0.0435 & 0.0434 & 0.0432 & 0.0455 & 0 & \cdots \\
10 & 0.0421 & 0.0452 & 0.0396 & 0.0450 & 0.0436 & 0.0436 & 0.0399 & 0.0410 & 0.0431 & 0.0436 & \cdots \\
 & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots &
\end{array} \qquad (1)$$

15 × 15 Matrix

We can conclude the m=7 is the block size since the entries are the biggest.
Now create 7 substrings. Also, use the alphabet frequenecy.

To retrieve the key, following steps are requried.
1. Iterate through 7 strings and count the frequency of each character
2. Compute Mg(Mutual index of concidence)
3. Retrieve the key from tracking down the Mg for each substring.

```
>
  M := 7:

  print("--- 2. Substring Generation & Frequency Count ---");
  yy := Array(1..M):
  nn_array := Array(1..M):
  fr := Array(1..M, 0..25, fill=0):

  for i to M do
      yy_seq := seq(yNew[j], j=i..n, M); # M=7 사용
      yy[i] := cat(yy_seq):
      nn_array[i] := nops([yy_seq]):

      for j to nn_array[i] do
          fr[i, code[yy[i][j]]] := fr[i, code[yy[i][j]]] + 1:
      od:
  od:

  print("--- 3. English Frequency Setup ---");
  ff := [
    .082, .015, .028, .043, .127, .022, .020, .061, .070, .002,
    .008, .040, .024,
    .067, .075, .019, .001, .060, .063, .091, .028, .010, .023,
    .001, .020, .001
  ]:
  f := Array(0..25):
  for i from 1 to 26 do
      f[i-1] := ff[i]:
  od:
```

"--- 2. Substring Generation & Frequency Count ---"

"--- 3. English Frequency Setup ---" (2)

Mutual Index of Coincidence measures how well a shifted ciphertext alphabet matches the expected frequency of the English letter.

For each substring, we test all possible shifts, here 26.

And for each shift, we calculate by multiplying the frequency of letter in English, and the frequency of letter in the ciphertext substring. The shift which gives the highest Mg value is most likely the correct key shift for that position, becuase it means the shifted ciphertext frequencies align best with normal English letter frequencies.

```
>
 print("--- 4. Calculating Mutual Index of Coincidence (Mg) ---");
 Mg := Array(1..M, 0..25):
 for i to M do
     for j from 0 to 25 do
         Mg[i,j] := add(f[k] * fr[i, (k+j) mod 26], k=0..25) /
 nn_array[i]:
     od:
 od:


 print("--- 5. Key Retrieval ---");
 K_indices := Array(0..M-1):
 K_letters := Array(0..M-1):

 for i to M do
     max_mg := 0.0:
     best_j := 0:

     for j from 0 to 25 do
         if Mg[i,j] > max_mg then
             max_mg := Mg[i,j]:
             best_j := j:
         end if:
     od:
```

```
    K_indices[i-1] := best_j:
    K_letters[i-1] := char[best_j]:
    printf("Substring %2d: Max Mg=%.4f (at shift j=%2d, key char=
'%s')\n", i, max_mg, best_j, char[best_j]);
od:

K_string := cat(seq(K_letters[i], i=0..M-1));
printf("==> Final Estimated Key (M=7): %s\n", K_string);

print("--- 6. Decryption ---");
print("Decrypting is just reverse of the Encryption");
p := cat(seq(char[(code[yNew[i]] - K_indices[(i-1) mod M]) mod 26],
i=1..n)):

print("Decrypted Plaintext:");
print(p);
```

"--- 4. Calculating Mutual Index of Coincidence (Mg) ---"

"--- 5. Key Retrieval ---"

```
Substring  1: Max Mg=0.0683 (at shift j=19, key char='T')
Substring  2: Max Mg=0.0713 (at shift j=25, key char='Z')
Substring  3: Max Mg=0.0678 (at shift j= 5, key char='F')
Substring  4: Max Mg=0.0661 (at shift j=19, key char='T')
Substring  5: Max Mg=0.0640 (at shift j=15, key char='P')
Substring  6: Max Mg=0.0622 (at shift j=12, key char='M')
Substring  7: Max Mg=0.0675 (at shift j= 2, key char='C')
```

$K_string := $ "TZFTPMC"

```
==> Final Estimated Key (M=7): TZFTPMC
```

"--- 6. Decryption ---"

"Decrypting is just reverse of the Encryption"

"Decrypted Plaintext:"

"ARTICLESCANBEDIVIDEDINTOSECTIONSBASEDONHOWMUCHCONTENTTHEREIS\ **(3)**
FOREACHAREATHEYREFERENCEASECTIONDEVOTEDTOMATHEMATICSISCOM\
MONHOWEVERITISOPTIONALWHENTHEREISONLYMATHEMATICALCONTENTP\
RESENTASIDEFROMREFERENCESOREXTERNALLINKSMATHEMATICALCONTEN\
TSHOULDCOMEBEFORECONTENTTHATDOESNOTDEALINPUREMATHEMATICSI\
NCLUDINGPRACTICALUSESINAPPLIEDFIELDSSUCHASSCIENCEANDEXTENDIN\
GTOCULTURALASSOCIATIONSANDTHEHISTORYOFSYMBOLSASSOCIATEDWIT\
HNUMBERSMATHEMATICALCOVERAGESHOULDOPENWITHSIMPLEMATHEMAT\
ICALPROPERTIESOFTHENUMBERANDLISTSOMESIMPLEASSOCIATEDPROPERTI\
ESWHETHERTHEYAREPRIMEORNOTABUNDANTETCTHENARTICLESHOULDM\
OVEONTODISCUSSDEEPERMATHEMATICSTHATISASSOCIATEDWITHTHENUMB\
ERSSUCHASPROPERTIESARISINGFROMBEINGAMEMBEROFSPECIFICCLASSESO\

FPRIMESORPROPERTIESTHATSTEMFROMRELATIONSHIPSWITHPARTICULARGE\
OMETRICFIGURESANYRELEVANTMATHEMATICALHISTORYOFTHENUMBERASI\
TRELATESTOTHESEPROPERTIESISALSOWORTHCOVERINGTHEADVISEFORASSE\
SSINGMATHEMATICALCONTENTSPECIFICALLYDISTINGUISHESBETWEENCONT\
ENTTHATISSIMPLEORBASICFORMULAICANDOFFTOPICREFERENCINGTHEONLI\
NEENCYCLOPEDIAOFINTEGERSEQUENCESOEISISLARGELYADMISSIBLEANDEN\
COURAGEDWITHINLIMITSCRITERIAGENERALLYFOLLOWSMERITSOFUNIQUEN\
ESSOFTHENUMBERSINQUESTIONWITHINTHESEQUENCESANDANYOTHERRELE\
VANTASSOCIATEDSEQUENCESASSESSINGDEEPERCONNECTIONSOFINTEGERS\
TOOTHERMATHEMATICALOBJECTSOFINTERESTISALSORELEVANTANDANIMP\
ORTANTPARTOFCOVERAGEOFMATHEMATICALPROPERTIESASCRIBEDTOTHEN\
UMBERSTHISGUIDELINEALSOEMPHASIZESPERWIKIPEDIAPOLICYTHATORIGIN\
ALRESEARCHINCLUSIVEOFSYNTHESISOFPUBLISHEDINFORMATIONISNOTPER\
MITTED"

Q3

Two time pad is created when the same encryption key is reused to encrypt two different plaintexts, resulting in two ciphertexts.

First step is to XOR the two ciphertexts together. This cancels out the reused key leaving with the XOR of two plaintexts.
C1=P1⊕K
C2=P2⊕K
C1⊕C2=(P1⊕K)⊕(P2⊕K)=P1⊕P2

Crib is a piece of text that you guess or know is part of one of the plaintexts. We can drag this crib across the Xor text by XORing at every possible position. When the crib is aligned a the correct position, the output of the XOR operation will reveal the corresponding part of the other plaintext.

The hint was given on the question, "Padding "s are used if the length of the plaintext was shorter than 650 bytes.

I could try entering several Padding
resulting of "__nd of the buffer."

This gave me a great hint to find the article about the buffer.
Testing more Paddings as a crib, I could get 'at produces more data could cause it to write past the end of the buffer.'
This was the description about the buffer overflow.

I searched up the article of buffer overflow in wikiepdia.
Could find the corresponding paragraph
"In programming and information security, a buffer overflow or buffer overrun is an anomaly whereby a program writes data to a buffer beyond the buffer's allocated memory, overwriting adjacent memory locations.

Buffers are areas of memory set aside to hold data, often while moving it from one section of a program to another, or between programs. Buffer overflows can often be triggered by malformed inputs; if one assumes all inputs will be smaller than a certain size and the buffer is created to be that size, then an anomalous transaction that produces more data could cause it to write past the end of the buffer."

Using this paragraph, it was possible to find the another plaintext, which was the paragraph of Vigenere-cipher article.

"The very first well-documented description of a polyalphabetic cipher was by Leon Battista Alberti around 1467 and used a metal cipher disk to switch between cipher alphabets. Alberti's system only switched alphabets after several words, and switches were indicated by writing the letter of the corresponding alphabet in the ciphertext. Later,

Johannes Trithemius, in his work Polygraphia (which was completed in manuscript form in 1508 but first published in 1518),[6] invented the tabula recta, a critical component of the Vigenere cipher.[7]
"
These were two plain texts included with "Padding "s used as the Padding.

Q4

Following the instruction step by step was enough to generate the working code.

The only function that was not described in detail was to decrypt the entire text part.
However, since having the other function completed,

Taking the full IV||ciphertext, chops it into 16byte blocks, and then recovers the message one block at a time by calling the block recovery function on each adjacent pair (previous and target). It collects the recovered plaintext blocks in order, making into the complete plaintext.
The important part is dealing with the padding bytes to return the original message.