University of Sussex

School of Engineering and Informatics

Project

Introduction to Computer Security

Module Convenor:

Dr. Imran U Khan

Candidate number:

164835

Table of Contents

Part A. 3

Task 1 3

Part B. 3

Task 2 13

Task 3 15

Task 4 19

Part C. 20

Task 5 20

Task 6 20

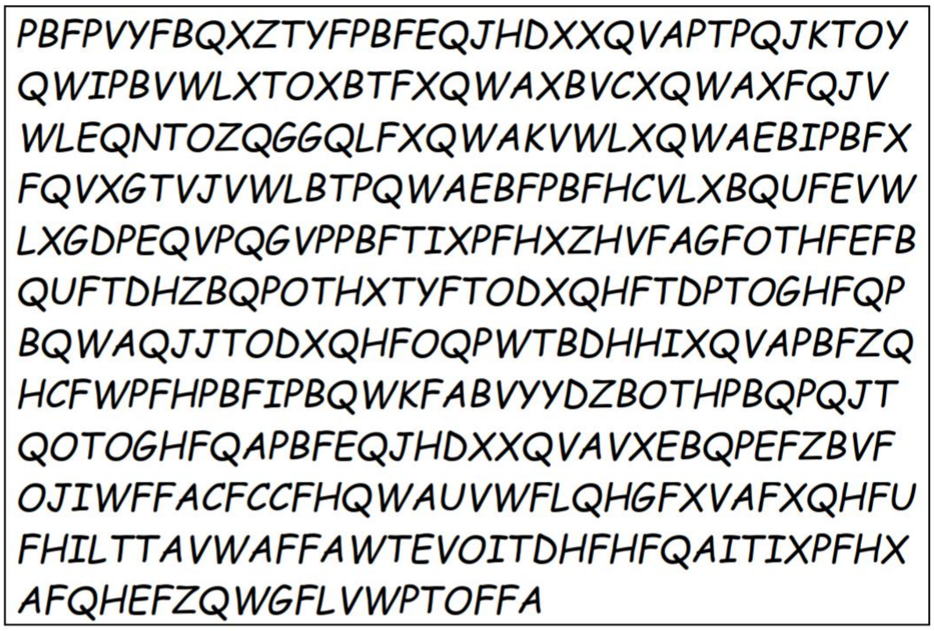
Task 7 24

Bibliography. 28

Part A

- Task 1

The brief about the ciphertext provided (Image 1) for decryption did not come with a key of any kind, neither did it contain any information about the algorithm that was used to encrypt the plaintext. Therefore, the most reasonable approach for cryptanalysis of said cipher text is frequency analysis.



```
PBFPVYFBQXZTYFPBFEQJHDXXQVAPTPQJKTOY
QWIPBVWLXTOXBTFXQWAXBVCXQWAXFQJV
WLEQNTOZQGGQLFXQWAKVWLXQWAEBIPBFX
FQVXGTVJVWLBTPQWAEBFPBFHCVLXBQUFEVW
LXGDPEQVPQGVPPBFTIXPFHXZHVFAGFOTHFEFB
QUFTDHZBQPOTHXTYFTODXQHFTDPTOGHFQP
BQWAQJJTODXQHFOQPWTBDHHIXQVAPBFZQ
HCFWPFHPBFIPBQWKFABVYYDZBOTHPBQPQJT
QOTOGHFQAPBFEQJHDXXQVAVXEBQPEFZBVF
OJIWFFACFCCFHQWAUVWFLQHGFXVAFXQHFU
FHILTTAVWAFFAWTEVOITDHFHFQAITIXPFHX
AFQHEFZQWGFLVWPTOFFA
```

Image 1. Provided ciphertext.

Frequency analysis is used to determine frequency of (groups of) letters, i.e. counting appearances of letters in the ciphertext. To perform frequency analysis on the given cipher text I used dCode frequency analysis tool (FAT) **[1]**.

Running FAT on the given encrypted text produced the following result:

Letter	Frequency Count	Frequency Percentage
F	51×	12.81%
Q	42×	10.55%
P	28×	7.04%
X	28×	7.04%
T	27×	6.78%
B	26×	6.53%
H	25×	6.28%
V	24×	6.03%
W	22×	5.53%
A	21×	5.28%
O	15×	3.77%
E	12×	3.02%
D	10×	2.51%
I	10×	2.51%
L	10×	2.51%
G	10×	2.51%
J	9×	2.26%
Z	8×	2.01%
Y	6×	1.51%
C	6×	1.51%
U	4×	1.01%
K	3×	0.75%
N	1×	0.25%

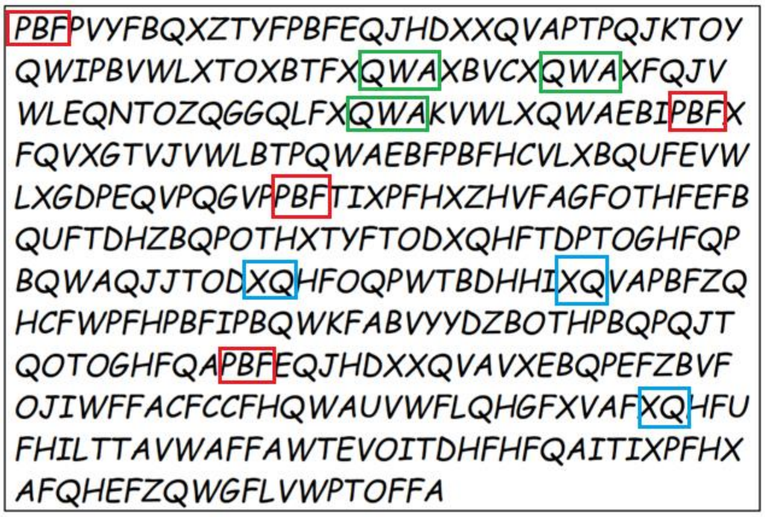
Table 1. Frequency of individual letters in cipher text **[1]**.

The table above shows the frequency of individual letters of the ciphertext. The first thing to take away from the initial analysis is that at least 3 letters of the alphabet aren't used in the plaintext, since the list above only has 23 entries. (assuming that a single letter in the cipher text represents one and only one letter of the alphabet). That, and the fact that the ciphertext is only 398 characters long (and is therefore too short of a sample) means that mapping the list in Table 2 will not produce a reasonable output.

Letter	Frequency Percentage
E	12.7%
T	9.1%
A	8.2%
O	7.5%
I	7.0%
N	6.7%
S	6.3%
H	6.1%
R	6.0%
D	4.3%
L	4.0%
C	2.8%
U	2.8%
M	2.4%
W	2.4%
F	2.2%
G	2.0%
Y	2.0%
P	1.9%
B	1.5%
V	1.0%
K	0.8%
J	0.2%
X	0.2%
Q	0.1%
Z	0.1%

Table 2. Letters by frequency of appearance in the English language **[1]**.

I therefore need more information to work with to be able to decrypt the ciphertext. Scanning though the cyphertext revealed repetitions of multiple trigrams and bigrams of letters (**Image 2**).



```
PBFPVYFBQXZTYFPBFEQJHDXXQVAPTPQJKTOY
QWIPBVWLXTOXBTFXQWAXBVCXQWAXFQJV
WLEQNTOZQGGQLFXQWAKVWLXQWAEBIPBFX
FQVXGTVJVWLBTPQWAEBFPBFHCVLXBQUFEVW
LXGDPEQVPQGVPPBFTIXPFHXZHVFAGFOTHFEFB
QUFTDHZBQPOTHXTYFTODXQHFTDPTOGHFQP
BQWAQJJTODXQHFOQPWTBDHHIXQVAPBFZQ
HCFWPFHPBFIPBQWKFABVYYDZBOTHPBQPQJT
QOTOGHFQAPBFEQJHDXXQVAVXEBQPEFZBVF
OJIWFFACFCCFHQWAUVWFLQHGFXVAFXQHFU
FHILTTAVWAFFAWTEVOITDHFHFQAITIXPFHX
AFQHEFZQWGFLVWPTOFFA
```

Image 2. Some of the bigrams/trigrams highlighted in the ciphertext.

To find out the exact number of n-grams the same tool can be used. Analysing for the cipher for bi-/tri-grams returned the following results:

Trigram	Count
PBF	8×
QWA	7×
VWL	5×
XQW	4×
WLX	3×
OTH	3×
XQV	3×
PFH	3×
QHF	3×
IPB	3×
HFQ	3×
FXQ	3×
BQP	3×
XQH	3×

Bigram	Count
PB	12×
QW	10×
XQ	10×
BF	9×
BQ	8×
HF	8×
VW	8×
WA	8×
TO	8×
FH	7×
FQ	6×
QH	6×
QJ	6×
WL	5×

Tables 3, 4. Frequencies of some of the bigrams and trigrams in the ciphertext **[1]**.

The top two trigrams appear 8 and 7 times in the ciphertext, which could indicate potential short words of the plain text. Some of the most common 3-letter words in the English language are ‘the’ and, well... ‘and’ **[2]**. If we look at the positions of the letters in ‘PBF’ and ‘the’, letters ‘F’ and ‘e’ are both at the top of Table 1 and Table 2 respectively. As for other letters, while their positions and percentage aren’t the same in two tables, they are comparable with no more than a few percent difference. This means that, we can try to map ‘PBF’ and ‘QWA’ to ‘the’ and ‘and’ respectively letter by letter and replace those letters in the ciphertext. For this task I used Python and its standard operations on text.

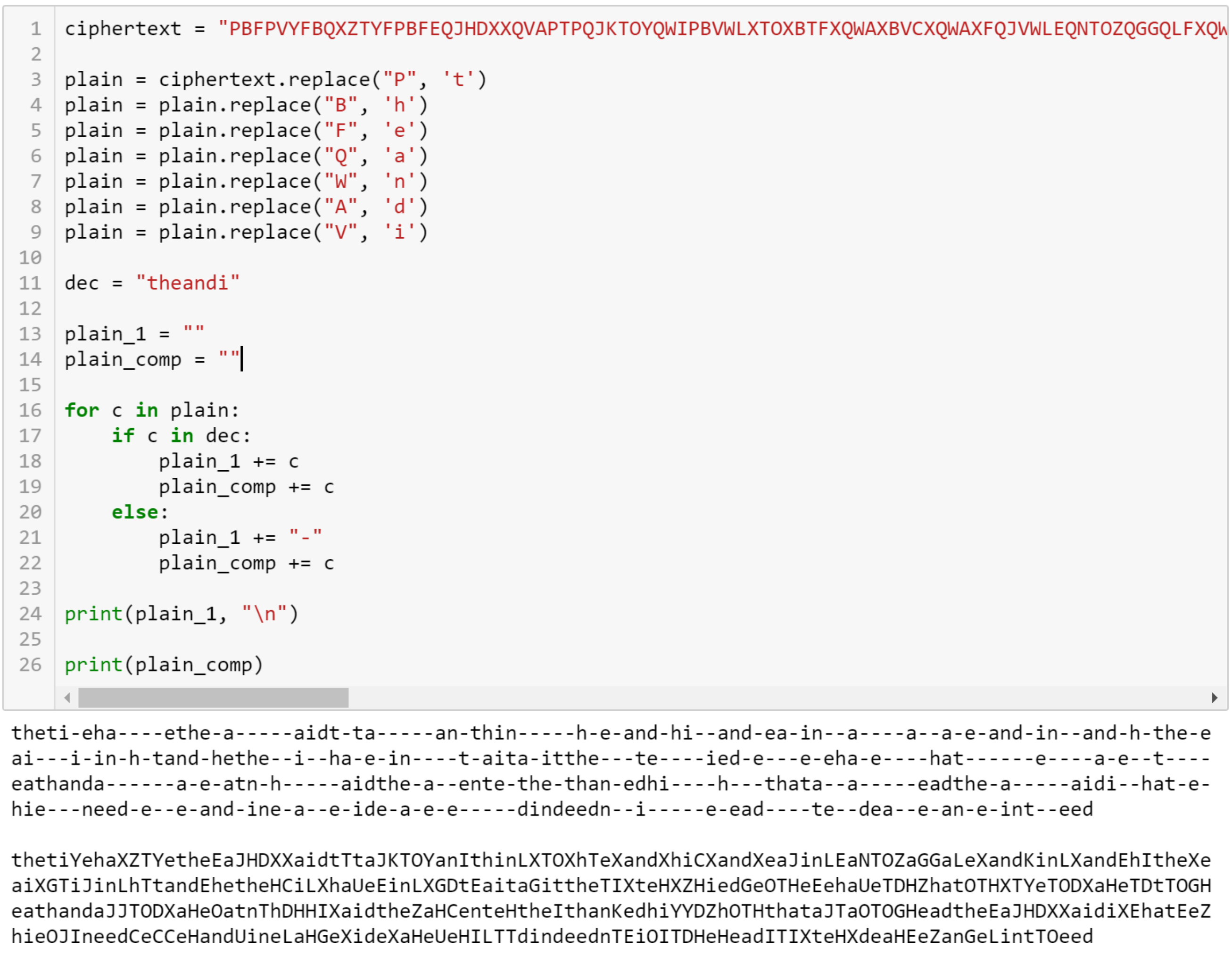
```python
text = "PBFPVYFBQXZTYFPBFEQJHDXXQVAPTPQJKTOYQWIPBVWLXTOXBTFXQWAXBVCXQWAXFQJVWLEQNTOZQGGQLFXQWAKVWLX

plain = text.replace("P", 't')
plain = plain.replace("B", 'h')
plain = plain.replace("F", 'e')
plain = plain.replace("Q", 'a')
plain = plain.replace("W", 'n')
plain = plain.replace("A", 'd')

dec = "theand"

plain_1 = ""
plain_comp = ""

for c in plain:
    if c in dec:
        plain_1 += c
        plain_comp += c
    else:
        plain_1 += "-"
        plain_comp += c

print(plain_1, "\n")

print(plain_comp)
```

```
thet--eha----ethe-a-----a-dt-ta-----an-th-n-----h-e-and-h---and-ea--n--a----a--a-e-and--n--and-h-the-e
a-------n-h-tand-hethe-----ha-e--n----t-a-ta--tthe---te-----ed-e---e-eha-e----hat------e----a-e--t----
eathanda------a-e-atn-h-----a-dthe-a--ente-the-than-edh-----h---thata--a-----eadthe-a-----a-d---hat-e-
h-e---need-e--e-and--ne-a--e--de-a-e-e-----d-ndeedn--------e-ead----te--dea--e-an-e--nt--eed

thetVYehaXZTYetheEaJHDXXaVdtTtaJKTOYanIthVnLXTOXhTeXandXhVCXandXeaJVnLEaNTOZaGGaLeXandKVnLXandEhItheXe
aVXGTVJVnLhTtandEhetheHCVLXhaUeEVnLXGDtEaVtaGVttheTIXteHXZHVedGeOTHeEehaUeTDHZhatOTHXTYeTODXaHeTDtTOGH
eathandaJJTODXaHeOatnThDHHIXaVdtheZaHCenteHtheIthanKedhVYYDZhOTHthataJTaOTOGHeadtheEaJHDXXaVdVXEhatEeZ
hVeOJIneedCeCCeHandUVneLaHGeXVdeXaHeUeHILTTdVndeednTEVOITDHeHeadITIXteHXdeaHEeZanGeLVntTOeed
```

The output below the code cell contains the current progress of decryption both with dashes for not yet decrypted characters (easier to read) and with undecrypted characters themselves for comparison.

The output is still not readable, but it can be analysed for potential longer words and further decryption (keeping in mind that some places that seem to contain a word could also be two words joined together, hence the trial-and-error nature of this analysis). For instance, the last line of the output contains a construct ‘--d-ndeedn--’, which potential contains the word ‘indeed’ in between two other words on ending with ‘d’ and other starting with ‘n’. The letter that position in ciphertext is

'V'. So for the next step I replace all 'V's in the ciphertext with 'i' (also their frequency percentages in Tables 1 and 2 has less than 1% difference, which makes it a reasonable substitution).

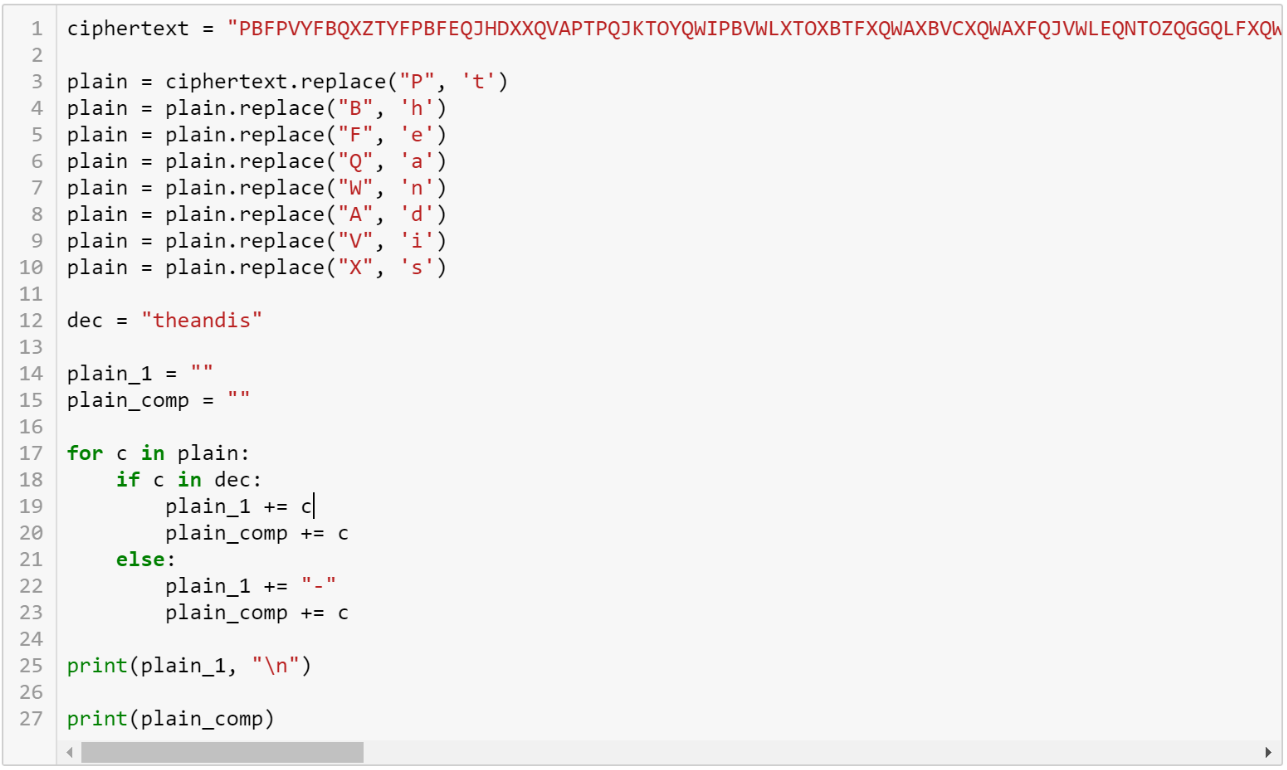


```python
ciphertext = "PBFPVYFBQXZTYFPBFEQJHDXXQVAPTPQJKTOYQWIPBVWLXTOXBTFXQWAXBVCXQWAXFQJVWLEQNTOZQGGQLFXQW

plain = ciphertext.replace("P", 't')
plain = plain.replace("B", 'h')
plain = plain.replace("F", 'e')
plain = plain.replace("Q", 'a')
plain = plain.replace("W", 'n')
plain = plain.replace("A", 'd')
plain = plain.replace("V", 'i')

dec = "theandi"

plain_1 = ""
plain_comp = ""

for c in plain:
    if c in dec:
        plain_1 += c
        plain_comp += c
    else:
        plain_1 += "-"
        plain_comp += c

print(plain_1, "\n")

print(plain_comp)
```

```
theti-eha----ethe-a-----aidt-ta-----an-thin-----h-e-and-hi--and-ea-in--a----a--a-e-and-in--and-h-the-e
ai---i-in-h-tand-hethe--i--ha-e-in----t-aita-itthe---te----ied-e---e-eha-e----hat------e----a-e--t----
eathanda------a-e-atn-h-----aidthe-a--ente-the-than-edhi----h---thata--a-----eadthe-a-----aidi--hat-e-
hie---need-e--e-and-ine-a--e-ide-a-e-e-----dindeedn--i-----e-ead----te--dea--e-an-e-int--eed

thetiYehaXZTYetheEaJHDXXaidtTtaJKTOYanIthinLXTOXhTeXandXhiCXandXeaJinLEaNTOZaGGaLeXandKinLXandEhItheXe
aiXGTiJinLhTtandEhetheHCiLXhaUeEinLXGDtEaitaGittheTIXteHXZHiedGeOTHeEehaUeTDHZhatOTHXTYeTODXaHeTDtTOGH
eathandaJJTODXaHeOatnThDHHIXaidtheZaHCenteHtheIthanKedhiYYDZhOTHthataJTaOTOGHeadtheEaJHDXXaidiXEhatEeZ
hieOJIneedCeCCeHandUineLaHGeXideXaHeUeHILTTdindeednTEiOITDHeHeadITIXteHXdeaHEeZanGeLintTOeed
```

The plain text is starting to take shape, but more detail is still needed to make it readable.

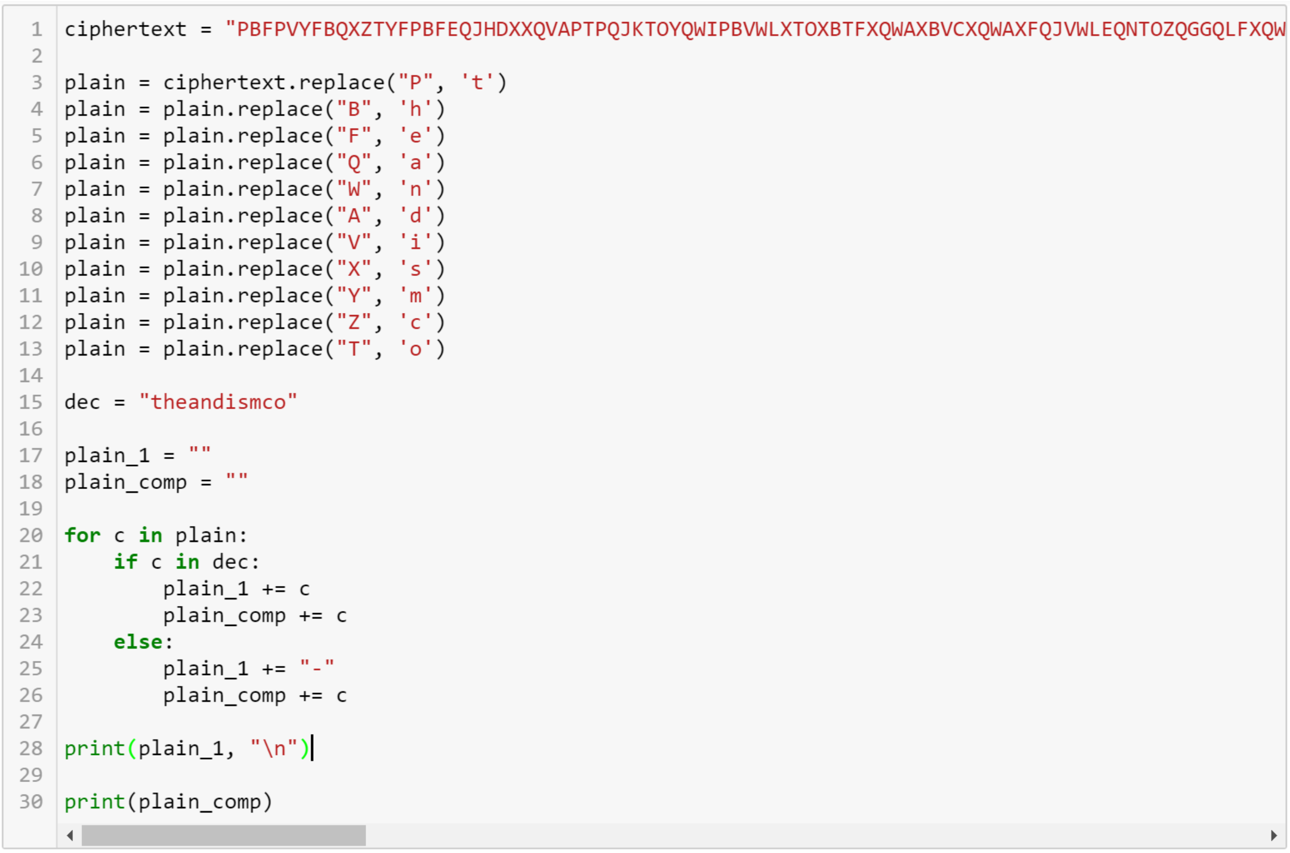
The next potential replacement is 'X' in the ciphertext. There's potential of words in multiple places in the ciphertext where 'X' occurs if replaces with 's' ('-haX-', '-Xaid-', '-Xea-'), and again the difference in frequencies between them in the ciphertext and English language respectively is less than 1% (see Tables 1 and 2). The result is presented below.



```python
ciphertext = "PBFPVYFBQXZTYFPBFEQJHDXXQVAPTPQJKTOYQWIPBVWLXTOXBTFXQWAXBVCXQWAXFQJVWLEQNTOZQGGQLFXQW

plain = ciphertext.replace("P", 't')
plain = plain.replace("B", 'h')
plain = plain.replace("F", 'e')
plain = plain.replace("Q", 'a')
plain = plain.replace("W", 'n')
plain = plain.replace("A", 'd')
plain = plain.replace("V", 'i')
plain = plain.replace("X", 's')

dec = "theandis"

plain_1 = ""
plain_comp = ""

for c in plain:
    if c in dec:
        plain_1 += c
        plain_comp += c
    else:
        plain_1 += "-"
        plain_comp += c

print(plain_1, "\n")

print(plain_comp)
```

```
theti-ehas---ethe-a---ssaidt-ta-----an-thin-s--sh-esandshi-sandsea-in--a----a--a-esand-in-sand-h-these
ais--i-in-h-tand-hethe--i-sha-e-in-s--t-aita-itthe--ste-s--ied-e---e-eha-e----hat---s--e---sa-e--t----
eathanda-----sa-e-atn-h----saidthe-a--ente-the-than-edhi----h---thata--a-----eadthe-a---ssaidis-hat-e-
hie---need-e--e-and-ine-a--esidesa-e-e-----dindeedn--i-----e-ead---ste-sdea--e-an-e-int--eed

thetiYehasZTYetheEaJHDssaidtTtaJKTOYanIthinLsTOshTesandshiCsandseaJinLEaNTOZaGGaLesandKinLsandEhIthese
aisGTiJinLhTtandEhetheHCiLshaUeEinLsGDtEaitaGittheTIsteHsZHiedGeOTHeEehaUeTDHZhatOTHsTYeTODsaHeTDtTOGH
eathandaJJTODsaHeOatnThDHHIsaidtheZaHCenteHtheIthanKedhiYYDZhOTHthataJTaOTOGHeadtheEaJHDssaidisEhatEeZ
hieOJIneedCeCCeHandUineLaHGesidesaHeUeHILTTdindeednTEiOITDHeHeadITIsteHsdeaHEeZanGeLintTOeed
```

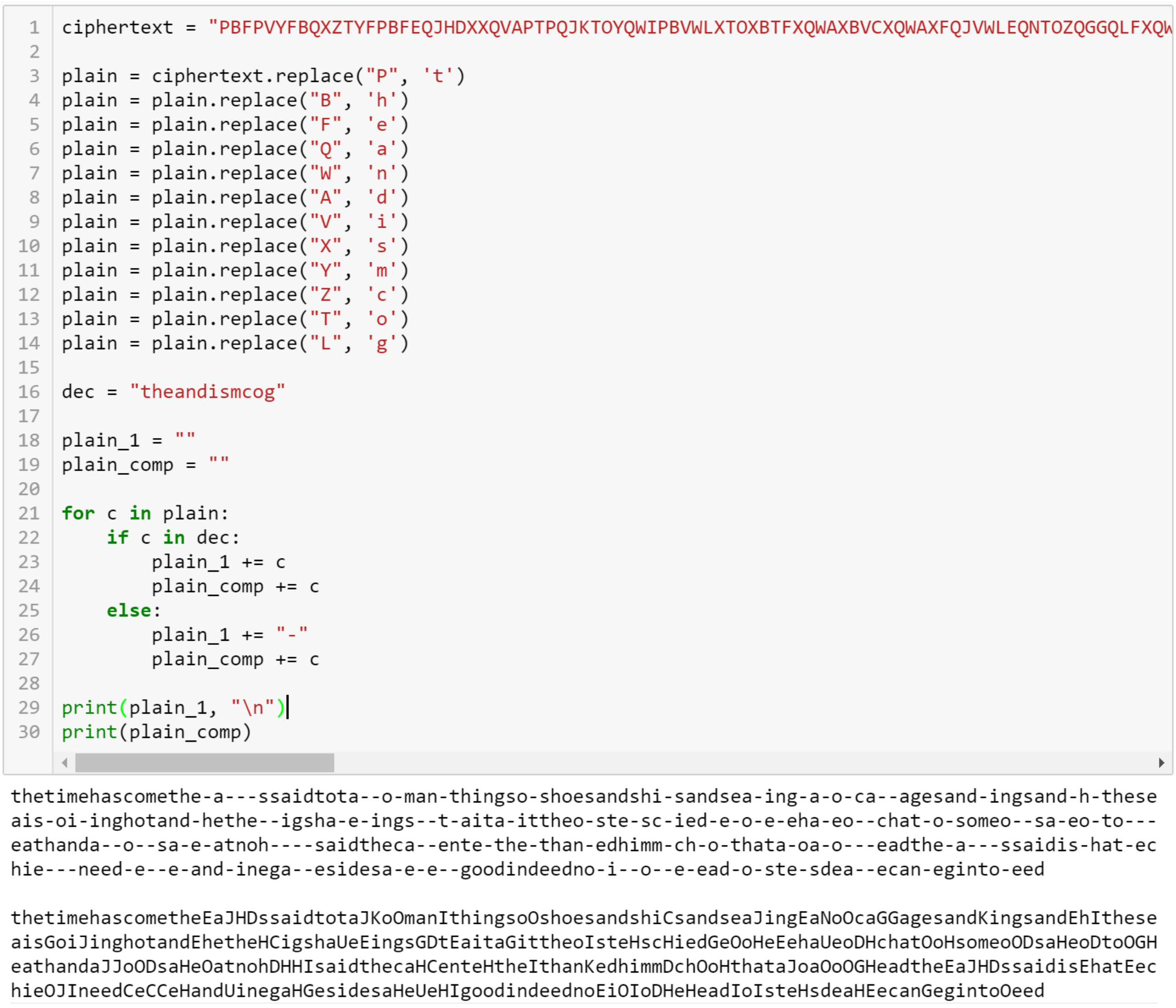
We can see that more words start emerging from decryption ('these', 'sea', 'said', 'than', etc.). Also, the very beginning of the text starts to look like the phrase 'the time has come'. For that, the following replacements must be made: 'Y'-'m', 'Z'-'c', 'T'-'o'. Again, referring to Tables 1 and 2, the frequencies of letters seem to be comparable.



```python
ciphertext = "PBFPVYFBQXZTYFPBFEQJHDXXQVAPTPQJKTOYQWIPBVWLXTOXBTFXQWAXBVCXQWAXFQJVWLEQNTOZQGGQLFXQW

plain = ciphertext.replace("P", 't')
plain = plain.replace("B", 'h')
plain = plain.replace("F", 'e')
plain = plain.replace("Q", 'a')
plain = plain.replace("W", 'n')
plain = plain.replace("A", 'd')
plain = plain.replace("V", 'i')
plain = plain.replace("X", 's')
plain = plain.replace("Y", 'm')
plain = plain.replace("Z", 'c')
plain = plain.replace("T", 'o')

dec = "theandismco"

plain_1 = ""
plain_comp = ""

for c in plain:
    if c in dec:
        plain_1 += c
        plain_comp += c
    else:
        plain_1 += "-"
        plain_comp += c

print(plain_1, "\n")

print(plain_comp)
```

```
thetimehascomethe-a---ssaidtota--o-man-thin-so-shoesandshi-sandsea-in--a-o-ca--a-esand-in-sand-h-these
ais-oi-in-hotand-hethe--i-sha-e-in-s--t-aita-ittheo-ste-sc-ied-e-o-e-eha-eo--chat-o-someo--sa-eo-to---
eathanda--o--sa-e-atnoh----saidtheca--ente-the-than-edhimm-ch-o-thata-oa-o---eadthe-a---ssaidis-hat-ec
hie---need-e--e-and-ine-a--esidesa-e-e---oodindeedno-i--o--e-ead-o-ste-sdea--ecan-e-into-eed

thetimehascometheEaJHDssaidtotaJKoOmanIthinLsoOshoesandshiCsandseaJinLEaNoOcaGGaLesandKinLsandEhIthese
aisGoiJinLhotandEhetheHCiLshaUeEinLsGDtEaitaGittheoIsteHscHiedGeOoHeEehaUeoDHchatOoHsomeoODsaHeoDtoOGH
eathandaJJoODsaHeOatnohDHHIsaidthecaHCenteHtheIthanKedhimmDchOoHthataJoaOoOGHeadtheEaJHDssaidisEhatEec
hieOJIneedCeCCeHandUineLaHGesidesaHeUeHILoodindeednoEiOIoDHeHeadIoIsteHsdeaHEecanGeLintoOeed
```

In the first line of the partially decrypted text there's another potential substitution: '-thin-so-' could be 'thinks', 'things' or 'think', 'thing' (which implies replacing 'L' with 'g' or 'k'), followed by a word starting with an 's'. But after trying both substitutions and analysing the rest of the text, replacing 'L' with 'g' makes more sense.



```python
ciphertext = "PBFPVYFBQXZTYFPBFEQJHDXXQVAPTPQJKTOYQWIPBVWLXTOXBTFXQWAXBVCXQWAXFQJVWLEQNTOZQGGQLFXQW

plain = ciphertext.replace("P", 't')
plain = plain.replace("B", 'h')
plain = plain.replace("F", 'e')
plain = plain.replace("Q", 'a')
plain = plain.replace("W", 'n')
plain = plain.replace("A", 'd')
plain = plain.replace("V", 'i')
plain = plain.replace("X", 's')
plain = plain.replace("Y", 'm')
plain = plain.replace("Z", 'c')
plain = plain.replace("T", 'o')
plain = plain.replace("L", 'g')

dec = "theandismcog"

plain_1 = ""
plain_comp = ""

for c in plain:
    if c in dec:
        plain_1 += c
        plain_comp += c
    else:
        plain_1 += "-"
        plain_comp += c

print(plain_1, "\n")
print(plain_comp)
```

```
thetimehascomethe-a---ssaidtota--o-man-thingso-shoesandshi-sandsea-ing-a-o-ca--agesand-ingsand-h-these
ais-oi-inghotand-hethe--igsha-e-ings--t-aita-ittheo-ste-sc-ied-e-o-e-eha-eo--chat-o-someo--sa-eo-to---
eathanda--o--sa-e-atnoh----saidtheca--ente-the-than-edhimm-ch-o-thata-oa-o---eadthe-a---ssaidis-hat-ec
hie---need-e--e-and-inega--esidesa-e-e--goodindeedno-i--o--e-ead-o-ste-sdea--ecan-eginto-eed

thetimehascometheEaJHDssaidtotaJKoOmanIthingsoOshoesandshiCsandseaJingEaNoOcaGGagesandKingsandEhIthese
aisGoiJinghotandEhetheHCigshaUeEingsGDtEaitaGittheoIsteHscHiedGeOoHeEehaUeoDHchatOoHsomeoODsaHeoDtoOGH
eathandaJJoODsaHeOatnohDHHIsaidthecaHCenteHtheIthanKedhimmDchOoHthataJoaOoOGHeadtheEaJHDssaidisEhatEec
hieOJIneedCeCCeHandUinegaHGesidesaHeUeHIgoodindeednoEiOIoDHeHeadIoIsteHsdeaHEecanGegintoOeed
```

On the same line there’s a potential word ‘total’ or combination of words ‘to talk’. The necessary substitutions are ‘J’-‘l’ and ‘K’-‘k’. Now while the difference in frequency between ‘J’ and ‘l’ is a little higher than between previous considered pairs of letters (about 1.5%), the percentages of ‘K’ and ‘k’ almost match in both tables (so no substitution needed).

```python
ciphertext = "PBFPVYFBQXZTYFPBFEQJHDXXQVAPTPQJKTOYQWIPBVWLXTOXBTFXQWAXBVCXQWAXFQJVWLEQNTOZQGGQLFXQW

plain = ciphertext.replace("P", 't')
plain = plain.replace("B", 'h')
plain = plain.replace("F", 'e')
plain = plain.replace("Q", 'a')
plain = plain.replace("W", 'n')
plain = plain.replace("A", 'd')
plain = plain.replace("V", 'i')
plain = plain.replace("X", 's')
plain = plain.replace("Y", 'm')
plain = plain.replace("Z", 'c')
plain = plain.replace("T", 'o')
plain = plain.replace("L", 'g')
plain = plain.replace("J", 'l')
plain = plain.replace("K", 'k')

dec = "theandismcoglk"

plain_1 = ""
plain_comp = ""

for c in plain:
    if c in dec:
        plain_1 += c
        plain_comp += c
    else:
        plain_1 += "-"
        plain_comp += c

print(plain_1, "\n")
print(plain_comp)
```

```
thetimehascomethe-al--ssaidtotalko-man-thingso-shoesandshi-sandsealing-a-o-ca--agesandkingsand-h-these
ais-oilinghotand-hethe--igsha-e-ings--t-aita-ittheo-ste-sc-ied-e-o-e-eha-eo--chat-o-someo--sa-eo-to---
eathandallo--sa-e-atnoh----saidtheca--ente-the-thankedhimm-ch-o-thataloa-o---eadthe-al--ssaidis-hat-ec
hie-l-need-e--e-and-inega--esidesa-e-e--goodindeedno-i--o--e-ead-o-ste-sdea--ecan-eginto-eed

thetimehascometheEalHDssaidtotalkoOmanIthingsoOshoesandshiCsandsealingEaNoOcaGGagesandkingsandEhIthese
aisGoilinghotandEhetheHCigshaUeEingsGDtEaitaGittheoIsteHscHiedGeOoHeEehaUeoDHchatOoHsomeoODsaHeoDtoOGH
eathandalloODsaHeOatnohDHHIsaidthecaHCenteHtheIthankedhimmDchOoHthataloaOoOGHeadtheEalHDssaidisEhatEec
hieOlIneedCeCCeHandUinegaHGesidesaHeUeHIgoodindeednoEiOIoDHeHeadIoIsteHsdeaHEecanGegintoOeed
```

In the same place, the possible construct is ‘to talk of’, also on the same line – ‘of shoes’, as well as ‘many things’. Therefore the next substitutions are ‘O’-‘f’ and ‘I’-y.

```
thetimehascomethe-al--ssaidtotalkofmanythingsofshoesandshi-sandsealing-a-ofca--agesandkingsand-hythese
ais-oilinghotand-hethe--igsha-e-ings--t-aita-ittheoyste-sc-ied-efo-e-eha-eo--chatfo-someof-sa-eo-tof--
eathandallof-sa-efatnoh---ysaidtheca--ente-theythankedhimm-chfo-thataloafof--eadthe-al--ssaidis-hat-ec
hieflyneed-e--e-and-inega--esidesa-e-e-ygoodindeedno-ifyo--e-eadyoyste-sdea--ecan-egintofeed

thetimehascometheEalHDssaidtotalkofmanythingsofshoesandshiCsandsealingEaNofcaGGagesandkingsandEhythese
aisGoilinghotandEhetheHCigshaUeEingsGDtEaitaGittheoysteHscHiedGefoHeEehaUeoDHchatfoHsomeofDsaHeoDtofGH
eathandallofDsaHefatnohDHHysaidthecaHCenteHtheythankedhimmDchfoHthataloafofGHeadtheEalHDssaidisEhatEec
hieflyneedCeCCeHandUinegaHGesidesaHeUeHygoodindeednoEifyoDHeHeadyoysteHsdeaHEecanGegintofeed
```

Applying the same logic, we can replace ‘C’ with ‘p’, and while there are other letters in Table 2 that have frequency close to frequency of ‘C’ in Table 1, ‘p’ makes the most sense, so one of the words we have decrypted as a result is ‘ships’, again in the first line.

```
thetimehascomethe-al--ssaidtotalkofmanythingsofshoesandshipsandsealing-a-ofca--agesandkingsand-hythese
ais-oilinghotand-hethe-pigsha-e-ings--t-aita-ittheoyste-sc-ied-efo-e-eha-eo--chatfo-someof-sa-eo-tof--
eathandallof-sa-efatnoh---ysaidtheca-pente-theythankedhimm-chfo-thataloafof--eadthe-al--ssaidis-hat-ec
hieflyneedpeppe-and-inega--esidesa-e-e-ygoodindeedno-ifyo--e-eadyoyste-sdea--ecan-egintofeed
```

```
thetimehascometheEalHDssaidtotalkofmanythingsofshoesandshipsandsealingEaNofcaGGagesandkingsandEhythese
aisGoilinghotandEhetheHpigshaUeEingsGDtEaitaGittheoysteHscHiedGefoHeEehaUeoDHchatfoHsomeofDsaHeoDtofGH
eathandallofDsaHefatnohDHHysaidthecaHpenteHtheythankedhimmDchfoHthataloafofGHeadtheEalHDssaidisEhatEec
hieflyneedpeppeHandUinegaHGesidesaHeUeHygoodindeednoEifyoDHeHeadyoysteHsdeaHEecanGegintofeed
```

Close to the beginning of the last line of the ciphertext, the hidden word (most likely ‘pepper’), requires replacing ‘H’ with ‘r’, and assuming it’s correct, the words that follow are ‘and vinegar besides’, so we can also replace ‘U’ with ‘v’ and ‘G’ with ‘b’.

```
thetimehascomethe-alr-ssaidtotalkofmanythingsofshoesandshipsandsealing-a-ofcabbagesandkingsand-hythese
aisboilinghotand-hetherpigshave-ingsb-t-aitabittheoysterscriedbefore-ehaveo-rchatforsomeof-sareo-tofbr
eathandallof-sarefatnoh-rrysaidthecarpentertheythankedhimm-chforthataloafofbreadthe-alr-ssaidis-hat-ec
hieflyneedpepperandvinegarbesidesareverygoodindeedno-ifyo-rereadyoystersdear-ecanbegintofeed
```

```
thetimehascometheEalrDssaidtotalkofmanythingsofshoesandshipsandsealingEaNofcabbagesandkingsandEhythese
aisboilinghotandEhetherpigshaveEingsbDtEaitabittheoysterscriedbeforeEehaveoDrchatforsomeofDsareoDtofbr
eathandallofDsarefatnohDrrysaidthecarpentertheythankedhimmDchforthataloafofbreadtheEalrDssaidisEhatEec
hieflyneedpepperandvinegarbesidesareverygoodindeednoEifyoDrereadyoystersdearEecanbegintofeed
```

The remaining undecrypted letters are ‘E’, ‘D’ and ‘N’. Simply by trying the remaining 6 letters of the alphabet, referring to the frequencies in Tables 1 and 2, and looking at the current decrypted portion of the text, I found the remaining hidden letters to be ‘w’, ‘u’ and ‘x’.

```
thetimehascomethewalrussaidtotalkofmanythingsofshoesandshipsandsealingwaxofcabbagesandkingsandwhythese
aisboilinghotandwhetherpigshavewingsbutwaitabittheoysterscriedbeforewehaveourchatforsomeofusareoutofbr
eathandallofusarefatnohurrysaidthecarpentertheythankedhimmuchforthataloafofbreadthewalrussaidiswhatwec
hieflyneedpepperandvinegarbesidesareverygoodindeednowifyourereadyoystersdearwecanbegintofeed
```

The plain text is therefore a piece of the poem by Lewis Carroll “The Walrus and the Carpenter”.

Part B

For this part the task is to come up with a new encryption algorithm (cipher) using substitution, transposition or a combination of both techniques.

- Task 2

The algorithm for my cipher will be presented with a set of steps needed for encryption and decryption followed by providing some more detail about the algorithm.

The available alphabet is defined as the string:

"abcdefghijklmnopqrstuvwxyz1234567890+-=*/^<>~!?,.;:\"'`_@#$£€₽&%\|()[]{} "

Encryption

1) Rotate the alphabet to the left by n (encryption key).

2) Split the alphabet into 9 clusters of 8 characters each.

3) Initialize a string where the cluster indices will be stored (this will form a part of the decryption key).

4) Iterate over plaintext character by character:

> **a)** for each character find the cluster it's contained in;
>
> **b)** note the index of the cluster and the index of the character in it;
>
> **c)** add the cluster index to the string initialized in **3**;
>
> **d)** add the number of dots (.) that matches the index of character (**b**) to the ciphertext;
>
> **e)** separate the encrypted character with '|'.

5) Separately note the string of cluster indices initialized in 3 and gradually built in **4b**.

6) Output ciphertext.

Decryption

1) Rotate the alphabet to the left by n (second part of decryption key).

2) Split the alphabet into 9 clusters of 8 characters each.

3) Iterate over ciphertext and over first part of decryption key (string of cluster indices produced by encryption):

> **a)** note the number of dots (.) before each separator (|);

b) note the current cluster index in the decryption key;

c) find the character in the array of clusters from **2** (with **b** being the index of cluster and **a** - the index of character in the cluster)

d) add the character produced by **c** to the plaintext.

4) Output plaintext.

The alphabet is defined as a string of 72 characters. The alphabet contains pretty much all the characters that can be found on an English (US/UK/EU) keyboard including Space (‘ ’). Although the alphabet can be altered, include less or more characters. I chose the length of 72 for a wider range of supported characters and for elegance of this implementation of the cipher (having uniform clusters). The cipher is not case sensitive; therefore, the plaintext can use both uppercase and lowercase characters, however after decryption the plaintext will only have lowercase characters.

When encrypting, the first order of business is to split the alphabet into clusters of characters and rotate it (for computation purposes those steps are done in different order in the algorithm). The number to rotate the alphabet by (**n**) is decided by the user and therefore acts as the encryption key for the algorithm. Rotating the alphabet adds a layer of complexity to the algorithm.

Splitting the alphabet produces 9 clusters 8 characters each. In computation, each cluster will have an index (0-8, but for simplicity I will use 1-9). Each character in the plaintext is therefore encrypted with a number of dot symbols (.). The number of dots is decided by the position of the character in one of the clusters of the alphabet. Each encrypted character (set of dots) is followed by a separator ‘|’.

Encryption also produces a part of the decryption key. The length of the decryption key matches plaintext’s length and is represented by a string of digits (1-9). Each digit represents the index of a cluster that contains the encrypted character. This string is stored separately and is used to build the decryption key along with the alphabet rotation index (also encryption key).

To decrypt the ciphertext, the alphabet again needs to be rotated (using the second part of decryption key) and split into clusters to match the alphabet that was used for encryption. Each set of dots in the ciphertext is matched with a digit from the first part of the decryption key. The digit and the number of dots in a set represent the cluster that a character is contained in and the position of that character in cluster (like pointing to an element in a 2-dimensional array). Applying these steps to each set of dots and digit eventually build the plaintext.

- Task 3

In this task I will illustrate the functionality of my cipher with an example.

Encryption

The string to encrypt: 'Toga!'.

Alphabet =

```
"abcdefghijklmnopqrstuvwxyz1234567890+-=*/^<>~!?,.;:\"'`_@#$£€₽&%\|()[]{} "
```

1) Rotate the alphabet by 5 (encryption key):

Alphabet =

```
"fghijklmnopqrstuvwxyz1234567890+-=*/^<>~!?,.;:\"'`_@#$£€₽&%\|()[]{} abcde"
```

2) Split the alphabet into clusters:

Clusters = array of strings of characters. Each cluster is 8 characters long.

```
(1) ['f g h i j k l m',
(2)  'n o p q r s t u',
(3)  'v w x y z 1 2 3',
(4)  '4 5 6 7 8 9 0 +',
(5)  '- = * / ^ < > ~',
(6)  '! ? , . ; : " '',
(7)  '` _ @ # $ £ € ₽',
(8)  '& % \ | ( ) [ ]',
(9)  '{ }   a b c d e']
```

3) Key = a string to store a part of the decryption key.

4) Iterating over the plaintext:

(First character)

a) The first character of plaintext is 't'. Looking at the clusters to find the character.

b) The number of the cluster where 't' is contained is 2. The position of the 't' in cluster is 7.

c) Key = Key + '2'. (Key = '2' so far).

d) Ciphertext = Ciphertext + '.......' (7 dots).

e) Ciphertext = Ciphertext + '|' (Plaintext = '.......|' so far).

(Second character)

a) The next character of the plaintext is 'o'. Looking at the clusters to find the character.

b) The number of the cluster where 'o' is contained is 2. The position of the 'o' in cluster is 2.

c) Key = Key + '2'. (Key = '22' so far).

d) Ciphertext = Ciphertext + '..' (2 dots).

e) Ciphertext = Ciphertext + '|' (Plaintext = '.......|..|' so far).

(Third character)

a) The next character of the plaintext is 'g'. Looking at the clusters to find the character.

b) The number of the cluster where 'g' is contained is 1. The position of the 'g' in cluster is 2.

c) Key = Key + '1'. (Key = '221' so far).

d) Ciphertext = Ciphertext + '..' (2 dots).

e) Ciphertext = Ciphertext + '|' (Plaintext = '.......|..|..|' so far).

(Fourth character)

a) The next character of the plaintext is 'a'. Looking at the clusters to find the character.

b) The number of the cluster where 'a' is contained is 9. The position of the 'a' in cluster is 4.

c) Key = Key + '9'. (Key = '2219' so far).

d) Ciphertext = Ciphertext + '....' (4 dots).

e) Ciphertext = Ciphertext + '|' (Plaintext = '.......|..|..|....|' so far).

(Last (fifth) character)

a) The next character of the plaintext is '!'. Looking at the clusters to find the character.

b) The number of the cluster where '!' is contained is 6. The position of the '!' in cluster is 1.

c) Key = Key + ‘1’. (Key = ‘22196’ so far).

d) Ciphertext = Ciphertext + ‘.’ (1 dot).

e) Ciphertext = Ciphertext + ‘|’ (Plaintext = ‘.......|..|..|....|.|’ so far).

6) Ciphertext = ‘.......|..|..|....|.|’.

Encryption completed.

First part of decryption key = ‘22196’.

Decryption

The string to decrypt: ‘.......|..|..|....|.|’

Alphabet =

"abcdefghijklmnopqrstuvwxyz1234567890+-=*/^<>~!?,.;:\"'`_@#$£€₽&%\|()[]{} "

Decryption key is made up of the string produced during encryption and the rotation of the alphabet. Therefore:

Decryption key = (‘22196’, 5)

1) Rotate the alphabet by 5 (second half of decryption key):

Alphabet =

"fghijklmnopqrstuvwxyz1234567890+-=*/^<>~!?,.;:\"'`_@#$£€₽&%\|()[]{} abcde"

2) Split the alphabet into clusters:

Clusters = array of strings of characters. Each cluster is 8 characters long.

```
(1) ['f g h i j k l m',
(2)  'n o p q r s t u',
(3)  'v w x y z 1 2 3',
(4)  '4 5 6 7 8 9 0 +',
(5)  '- = * / ^ < > ~',
(6)  '! ? , . ; : " '',
(7)  '` _ @ # $ £ € ₽',
(8)  '& % \ | ( ) [ ]',
(9)  '{ }   a b c d e']
```

3) Going over ciphertext and first half of decryption key.

(Set of dots before the first separator and the first number in the decryption key)

a) The number of dots before first separator is 7 (‘.......|..|..|....|.|’).

b) The first number in the first part of decryption key is 2 ('22196').

c) The character in cluster 2 at position 7 is 't'.

d) Plaintext = Plaintext + 't' (Plaintext = 't' so far).

(Set of dots before the second separator and the second number in the decryption key)

a) The number of dots before first separator is 2 ('.......|..|..|....|.|').

b) The first number in the first part of decryption key is 2 ('22196').

c) The character in cluster 2 at position 2 is 'o'.

d) Plaintext = Plaintext + 'o' (Plaintext = 'to' so far).

(Set of dots before the third separator and the third number in the decryption key)

a) The number of dots before first separator is 2 ('.......|..|..|....|.|').

b) The third number in the first part of decryption key is 2 ('22196').

c) The character in cluster 1 at position 2 is 'g'.

d) Plaintext = Plaintext + 'g' (Plaintext = 'tog' so far).

(Set of dots before the fourth separator and the fourth number in the decryption key)

a) The number of dots before first separator is 4 ('.......|..|..|....|.|').

b) The third number in the first part of decryption key is 9 ('22196').

c) The character in cluster 9 at position 4 is 'a'.

d) Plaintext = Plaintext + 'g' (Plaintext = 'toga' so far).

(Set of dots before the last separator and the last number in the decryption key)

a) The number of dots before first separator is 1 ('.......|..|..|....|.|').

b) The third number in the first part of decryption key is 6 ('22196').

c) The character in cluster 6 at position 1 is '!'.

d) Plaintext = Plaintext + '!' (Plaintext = 'toga!' so far).

4) Plaintext = 'toga!'.

Decryption completed.

The example of my cipher's workings is now complete. Should be noted that rotating the alphabet by a different number of characters would completely change both the ciphertext and the decryption key. Also, the alphabet used for decryption must match the one used for encryption.

- Task 4

To implement my encryption algorithm, I used Python. Not to clutter this paper, I included a Python file with my cipher implemented (my_cipher.py) in the project folder.

Part C

- Task 5

One immediately noticeable issue with applications passwords policy is in the Users table (dvd.mbd file). The passwords (along with emails and other data) are clearly stored as plaintext and refer directly to the users they belong to. This puts not only the credibility of the application in risk, but more importantly, user's security. Besides, the application clearly allows simple passwords to be used (like the word 'password'), that also don't include any special characters. So, one immediate suggestion is for the application to have a stricter password policy, e.g. only allow passwords of adequate length (more than 8 characters long) and that include at least one special character, one upper case letter, one lower case letter and a digit. There could also be a collection of words that are banned from being used as (part of) a password.

Looking through other files, I couldn't find any mechanism of password encryption, which is totally unacceptable by today's technology and security standards. Therefore, there must be some level of encryption in place for user passwords, maybe by using one of the common encryption algorithms for passwords like AES or RSA.

The file signup.aspx.cs contains the password creation mechanism. Passwords for new users are apparently generated randomly by the application using some list of words (essentially a dictionary) and sends the passwords (again, unencrypted) to the users by email. This, along with the fact that many users may not bother to change their password later (which application allows to do) makes it an easy target for attackers. Suggestions here are the same as before – for the application to have an encryption policy in place and encourage users to create more complex passwords following the guide suggested before, since at the end of the day, it is both responsibility of the app and the users to keep personal user data secure.

Other suggestion include, implementing two-factor authentication or use advanced authentication methods offered by modern devices (fingerprints, facial recognition, iris scanning, etc.)

- Task 6

Data Encryption Standard (DES), Triple DES (3DES) and Advances Encryption Standard are all symmetric key encryption algorithms that had progressively replaced each other (in that order) over time due to inefficiency or vulnerabilities. In this section I will discuss the major architectural differences between the three ciphers.

Length of the key

Being an implementation of Feistel Cipher, the effective key length in DES is 56 bits (+8 serving the function of check bits). The key is used then to encrypt data through 16 rounds of encryption + initial and final permutations (although for each round a separate 48-bit key is generated from 56-bit cipher key). Over time, DES has been proven to be not only slow, but also, due to rapid developments in computation power and relatively short key length, vulnerable **[3]**.

Triple DES was an iterative upgrade from DES (since replacing a widely adopted encryption algorithm would take a lot of time and money). The implemented changes were the way DES was used and the length of the key. Triple DES effectively used either two 56-bit long keys (total of 112-bit) or three 56-bit keys (total key length is 168 bits). Increasing the length of the key and other internal changes lead to adequate security of 3DES but at the same time made it even slower than classic DES **[4]**.

AES nowadays is the more popular and widely used symmetric cipher that is at least 6 times faster than DES. The length of the key for AES may vary depending on the number of rounds in encryption process. The possible key lengths are 128, 192 and 256 bits. The architecture of AES lead not only to improved speed but also stronger security **[5]**.

Size of a single block for encryption

All three ciphers encrypt a chunk or block of data at a time. The both DES and 3DES have the block size of 64 bits, however the internal workings on a block is a different (more on that further in the task) **[3, 4]**. AES on the other hand increases the supported block size to 128 bits, so double the size of a block in DES and 3DES **[5]**.

Number of rounds during encryption

DES encrypts its blocks through 16 rounds of Feistel structure **[3]**. 3DES effectively triples the number of rounds to 48 **[4]**, which caused one of the performance hits for triple DES (speed). In AES the number of rounds that data is encrypted through will vary depending on the size of the key: for 128, 192 and 256 bits long keys AES has 10, 12 and 14 rounds respectively **[5]**.

Overall internal mechanism of encryption

DES uses 16 round Feistel structure **[3]**. Initial and final permutations also take place and are essentially the inverses of each other. Each round applies a 48-bit key to the rightmost 32 bits of the input and produces output of the same length. Because the input size (32 bits) is different from key size (48 bits) the input must be extended in an Expansion Permutation Box.

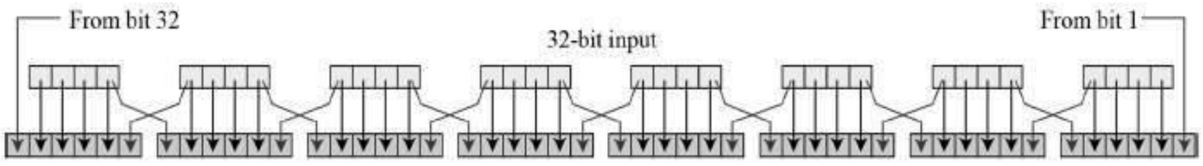


Image 3. Example of Expansion Permutation Box **[3]**.

Then XOR operation takes place on the expanded section and the key of the current round. After that Substitution boxes perform the actual mixing of data. DES makes use of 8 such boxes, each box has 6 bits of input and 4 bits of output.

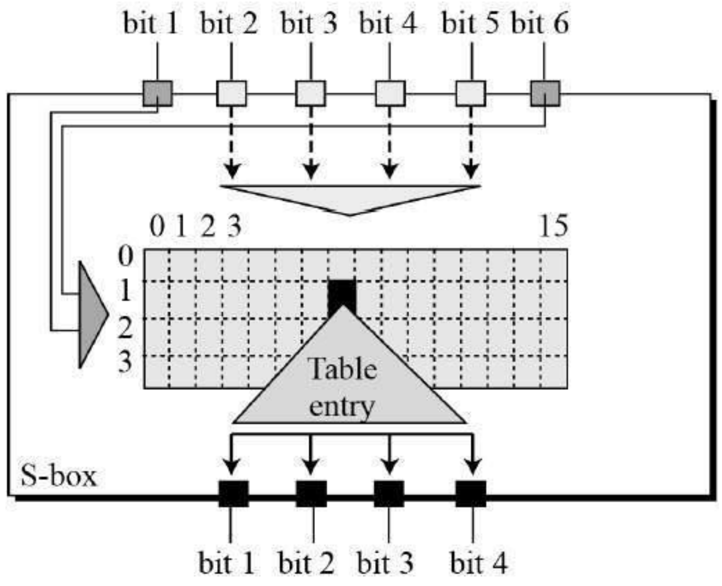
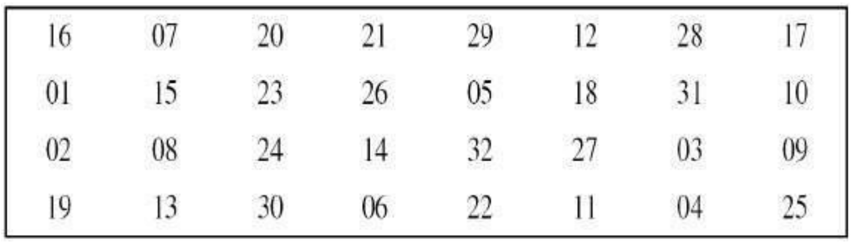


Image 4. Example of Substitution Box **[3]**.

Then there's Straight Permutation, that has its own specific rule in place (shown below).



16	07	20	21	29	12	28	17
01	15	23	26	05	18	31	10
02	08	24	14	32	27	03	09
19	13	30	06	22	11	04	25

Image 5. Straight Permutation rule **[3]**.

Triple DES takes the process that is used in DES and 'triples' it: the 64-bit block (same as DES) is encrypted using a 56 bits long key, then decrypted with another 56-bit key, and finally encrypted again using yet another 56-bit key (hence the total 168 bits size of the key). There's also a variant of Triple DES that uses has 2-part key (56 + 56 = 112 bits long key). The process is the same in this case, but during final encryption the first key is used as opposed to a completely new one in 3-key Triple DES **[4]**.

AES is iterative (rather than Feistel like DES) and is uses substitution-permutation network. Also, unlike DES and 3DES, it operates with bytes (128 bits long block is treated like 16 bytes long block). The bytes create a 4x4 matrix for processing. The number of rounds, as mentioned before, depends on the length of the key. Encryption is a combination of 4 processes: byte substitution, which substitutes the input bytes by a value in a Substitution Box and stores the result in a 4x4 matrix; shifting rows of the matrix to the left by more bytes with each row (1st row not shifted, 2nd, 3rd and 4th shifted by 1, 2 and 3 respectively); mixing columns with a special mathematical function; XOR performed between the final (16 bytes, 128 bits) matrix and the key (also 16 bytes, 128 bits) **[5]**.

The overall performance comparison of the three ciphers based on memory usage and security is shown in the diagram below.

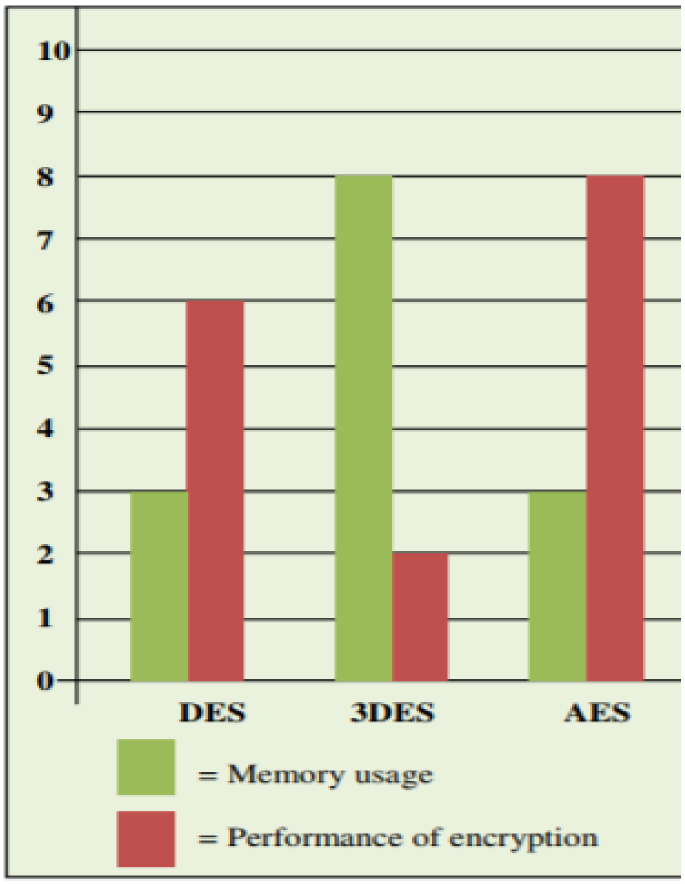


Image 6. Comparison of the ciphers **[6]**.

- Task 7

General Data Protection Regulation (GDPR) – is a set of rules, a regulation in the European Union law that concerns privacy and protection of data for EU citizens. Its primary goal is to provide individuals more control over the use of their personal data as well as to simplify the regulatory environment for businesses to allow everyone (citizens and businesses) can benefit from advantages of digital economy.

Data Protection Directive (DPD) – a predecessor for GDPR, an EU directive that regulates processing of personal data.

GDPR implements a number of major changes from DPD that concern personal data, individual rights, security, penalties and others that are discussed in this section.

Definition of personal data

GDPR is supposed takes into account changes in technology and how personal data is collected by organizations. This is beneficial for users in terms of privacy but can harm modern marketing and sales practices, because building a profile of some individual through his/her preferences, browsing and purchase history etc., is only possible with the said individual’s consent.

While in DPD personal data includes names, IDs (banking, social security, etc.), photos, emails, phone numbers, GDPR extends the range of information by labelling as personal data not only everything defined in DPD, but also any other data that can be used to identify a person (IPs, device IDs, geolocation, biometrics, as well as person’s physical, psychological, genetic, mental, economic, cultural, or social traits).

Individual rights

GDPR represents the next step in privacy considerations by requiring a formal agreement from individuals for processing of their personal data, and the implications of such agreement must be concise, but at the same time informative and unambiguous. In addition, its unacceptable to ask consumers to agree to terms of a contract in exchange for their consent for personal data to be used and different types of data require separate consent.

Users preserve the right to access their personal data, i.e. the ability to request information from data controllers about how, where and why personal data is used. Such request must be satisfied with no charge indefinitely by data controllers by providing the said information along with a digital copy of user’s personal data with.

Users have the right to request the transfer their data (e.g., from one service provider to another) and to be forgotten, i.e. erasing personal data by data controllers, cease any further use if data, and/or halt the use of data by third-parties should the user initiate such a request.

Data processors and data controllers

One of the differences between DPD and GDPR is the regulation of data processors by the GDPR. Data controllers and data processors are both responsible to adhere to new policies, i.e. if an organization shares data or its analysis with a third party and/or processes data on behalf of a third party, both sides are obligated to abide by the GDPR and both are amendable for violations.

In DPD, only data controllers were accountable for violations of existing rules. Under GDPR, on the other hand, data controllers and data processors are bind with a contract over processing of any personal data.

In GDPR a data controller is defined as “natural legal person, public authority, agency or other body, which determines the purposes and means of the processing of personal data," while a data processor is "natural legal person, public authority, agency or other body, which processes data on behalf of the controller."

Information governance and security

GDPR obligates organizations to actively track the storage and use of data by implementing risk management tools and building privacy and security into their operations by design. For privacy, i.e. considering the privacy of collected data at every step in the development of business concepts and discarding of personal data by data controllers when it's no longer in use; for security i.e. conducting impact assessments by organizations for automated data processing activities and large-scale processing of certain kinds of data.

Breach of data and penalties

Under GDPR data breaches must be reported by organizations (both to the owner of compromised data and supervising authority) within 72 hours of the breach. The notification should include: the nature of the breach; the categories and number of impacted individuals (approximately); and the contact information data protection office; explanation of the possible consequences of the breach and measures undertaken by data controller to address and ease the effect of the breach. The supervisory authority then must evaluate the compromised data and the security measures on the occurrence of the breach to assess the effect and ensure future consent from prospective users.

Under DPD, members of the EU were free to introduce their own laws concerning data breach notifications, which means that when a breach took place, companies had to conduct research and ensure compliance with each member state.

Violations such as lack of consent to process data, or not implementing privacy by design (see previous section) could result in a charge either of 4% of their turnover or €20 million (whatever is higher). Less impactful violations (disordered records, not notifying the supervising authority) can result in a lesser charge of 2% of global turnover.

Geographical impact

The fact that GDPR makes use of digital personal and that it applies to the processing of personal data of individuals who reside in the EU even if data controller or processor was not established in the EU, expands its the geographical reach far beyond DPD and effectively makes GDPR a worldwide law.

Other notable changes

- The minimum age of individuals whose data is liable for collection is raised from 13 to 16.

- There exists a single national office for dealing with complaints.

- Larder data controllers must assign a data protection officer.

Bibliography

[1] dCode, (2018). Frequency Analysis (advanced), Frequency Analysis. [Online]. Available at: https://www.dcode.fr/frequency-analysis [Accessed 28Nov 2018]

[2] Practical Cryptography, (2012). English Letter Frequencies. [Online]. Available at: http://practicalcryptography.com/cryptanalysis/letter-frequencies-various-languages/english-letter-frequencies/ [Accessed 28 Nov 2018]

[3] TutorialsPoint, (2018). Data Encryption Standard. [Online]. Available at: https://www.tutorialspoint.com/cryptography/data_encryption_standard.htm [Accessed 2 Dec 2018]

[4] TutorialsPoint, (2018). Triple Data Encryption Standard. [Online]. Available at: https://www.tutorialspoint.com/cryptography/triple_des.htm [Accessed 3 Dec 2018]

[5] TutorialsPoint, (2018). Advanced Encryption Standard. [Online]. Available at: https://www.tutorialspoint.com/cryptography/advanced_encryption_standard.htm [Accessed 3 Dec 2018]

[6] Mushtaque, M. A., (2014). Comparative analysis on different parameters of encryption algorithms for information security. International Journal of Computer Sciences and Engineering, 2(4). [Online]. Available at: http://www.ijcseonline.org/pub_paper/IJCSE-00187.pdf [Accessed 4 Dec 2018]

[7] Voigt, P., & Von dem Bussche, A. (2017). The EU General Data Protection Regulation (GDPR) (Vol. 18). Springer.

[8] Beaumont, S., (2018). The Data Protection Directive versus the GDPR: Understanding key changes. [Online]. Available at: https://www.synopsys.com/blogs/software-security/dpd-vs-gdpr-key-changes/ [Accessed 5 Dec 2018]