Computer Security – CSCI_6531 – Homework 1

Namana Y Tarikere – G21372717

1. **BASIC CRYPTANALYSIS**

For the following two encryption problems, use a brute force method (it may be an "informed" brute force method, using features of the plaintext you infer) or frequency analysis. Show all of your work. It is my expectation that you solve the problems yourself and describe the method used. Any answer similar to "I put the ciphertext into a prompt on a web site" will result in no credit.

1.1 Decryption (5 pts)

EUVk4NaRo/R_cR_kX\``Vkw0V_V\a`Z`o[`cRRY\]YR[UVRRL}Nt{kd\^ReV^RdkRYd\oX_\h[o NdkeUVkaVXkX\``VwoVdkRkd]VPZRdk`SoT`\dRoR_QVZZPoa`kd\faYRc[o.f`e_RYZN}k:aoVd kRkUVdaZ[TaZcVk]NcTVwoTcRjkSVcQoaYNekZ`oZ``eYjkeRc_V`e_ZN]kR[UkZ`o[`aoP]\dR]fo _VYRaVQoa`k`aYR ckVeeN_aoZVZSRc`o\WkeUVkdbSSRZZYjk2[dRcV_NVy

The above is ciphertext, encrypted with a variation of the Vigenère cipher that works on UTF-8 values for the characters. Recover the plaintext.

Solution:

The decrypted plaintext for the above Vigenère ciphertext is:

The~Cape~Barren~goose~(Cereopsis~novaehollandiae[2])~sometimes~also~known~as~the~pig~goose~is~a~species~of~goose~endemic~to~southern~Australia~It~is~a~distinctive~large~grey~bird~that~is~mostly~terrestrial~and~is~not~closely~related~to~other~extant~members~of~the~subfamily~Anserinae

----------x----------x----------x----------

1.2 Analysis (20 pts)

Describe how you recovered the plaintext for the above. What were the challenges? How long did your method take? What features of the cipher made the method easier or harder to execute successfully? Include any code you wrote. You may use comments in your code as part of your description.

Solution:

I used the Kasiski model and frequency analysis methods in my python code to decrypt the given Vigenère ciphertext by following the below steps:

a. Firstly, I analysed the most repetitive trigrams (patterns of 3) from the ciphertext [2] by running a loop I calculated the occurrence of each pattern and added the count to a dictionary. The result of the most repeated patterns of trigrams along with their frequency is:
{3: {'Z`o', 'dkR', 'UVk'}, 2: {'a`k', 'Vwo', '`e_', 'Qoa', 'eUV', 'keU', 'X\\`', 'Yjk', 'kd\\', '`o[', 'oVd', 'Rdk', 'kRk', '\\dR', '``V', 'R[U', 'kZ`', 'kX\\', '\\``', 'o[`', 'oa`', 'Vdk', 'YRc', 'aYR'}}

Here the patterns {'Z`o', 'dkR', 'UVk'} are repeated 3 times each, so I considered them to find the index of occurrence of each of the pattern and calculated the difference in index between each of the occurrences in the ciphertext for understanding the key length [1] and splitting the text.

b. To find the indices of the trigrams for each of its occurrence, I wrote the second part of the code to run a 'for' loop on the dictionary and appended the result as shown:

{'UVk': [1, 78, 263], 'dkR': [61, 93, 148], 'Z`o': [29, 186, 212]}

Here, the trigram 'UVk' is occurring at 1st, 78th and 263rd index of the cipher text, trigram 'dkR' is occurring at 61st, 93rd and 148th index of the cipher text and trigram 'Z`o' is occurring at 29th, 186th and 212nd index of the cipher text. The difference between these indices gives the length after which the trigram is repeating in the ciphertext as shown below.

{'UVk': [1, 78, 263], 'dkR': [61, 93, 148], 'Z`o': [29, 186, 212]}
UVk: [77, 262, 185]
dkR: [32, 87, 55]
Z`o: [157, 183, 26]

Next, I am calculating all the factors of these differences in the indices to find the possible key length.

Factors of 77: 1, 7, 11, 17
Factors of 262: 1, 2, 131, 262
Factors of 185: 1, 5, 37, 185
Factors of 32: 1, 2, 4, 8, 16, 32
Factors of 87: 1, 3, 29, 87
Factors of 55: 1, 5, 11, 55
Factors of 157: 1, 157
Factors of 183: 1, 3, 61, 183
Factors of 26: 1, 2, 13, 26

Here, possible key length will be the maximum value of the most repeated factors. Factor 2 is repeated thrice and factor 3 is repeated twice as shown above. Since factor 2 is repeating the most, I considered the possible **key length to be 'two'** based on the Kasiski method of solving Vigenère cipher [1].

c. In step 3, after finding the key length to be '2', I split the entire cipher text by length of 2 for frequency analysis [2]. The result after split will be:

EU Vk 4N aR o/ R_ cR _k X\ `` Vk w0 V_ V `Z `o [` cR RY \] YR [U
VR RL }N t{ kd \^ Re V^ Rd kR Yd \o X_ \h [o Nd ke UV ka VX kX \`
`V wo Vd kR kd ]V PZ Rd k` So T` \d Ro R_ QV ZZ Po a` kd a YR c[o.
f` e_ RY ZN }k :a oV dk Rk UV da Z[Ta Zc Vk]N cT Vw oT cR jk SV
cQ oa YN ek Z` oZ `` eY jk eR c_ V` e_ ZN ]k R[ Uk Z` o[`a oP]\ dR
]f o_ VY Ra VQ oa `k `a YR ck Ve eN _a oZ VZ SR c` o\ Wk eU Vk db
SS RZ ZY jk 2[dR cV _N Vy

d. Next, I computed the frequency analysis of each letter on the split text to find the count of each letter starting at either index 0 or 1 and mapped the count to the respective letters along with their indices. Frequency analysis of split letters for respective index points is shown below:

{'E': {0: 1}, 'U': {1: 3, 0: 3}, 'V': {0: 17, 1: 8}, 'k': {1: 26}, '4': {0: 1}, 'N': {1: 8, 0: 1}, 'a': {0: 3, 1: 11}, 'R': {1: 16, 0: 11}, 'o': {0: 19}, '/': {1: 1}, '_': {1: 6, 0: 5}, 'c': {0: 9, 1: 2}, 'X': {0: 3, 1: 1}, '\\': {1: 10}, '`': {0: 10, 1: 10}, 'w': {0: 1, 1: 2}, '0': {1: 1}, 'Z': {0: 10, 1: 5}, '[': {1: 8}, 'Y': {0: 4, 1: 5}, ']': {1: 2, 0: 4}, 'L': {0: 1}, '}': {1: 1, 0: 1}, 't': {1: 1}, '{': {0: 1}, 'd': {0: 14}, '^': {0: 2}, 'e': {0: 9, 1: 1}, 'h': {0: 1}, 'P': {1: 3}, 'S': {1: 2, 0: 3}, 'T': {1: 3, 0: 1}, 'Q': {1: 3}, 'f': {0: 2, 1: 1}, '.': {1: 1}, ':': {0: 1}, 'j': {0: 3}, 'W': {0: 1}, 'b': {1: 1}, '2': {0: 1}, 'y': {1: 1}}

Here, I observed that letter **'o' is occurring 17 times at 0th index** of the split text and letter **'k' is occurring 26 times at the 1st index** of the split text making them the possible keys of length 'two' for the encrypted text as described in the Kasiski's model [1]. **Therefore, I consider the key length to be 2 and possible key for decryption to be 'ok'**.

e. Now, I computed an algorithm to decrypt the text using the found key 'ok'. Since the problem definition mentioned that "the encrypted text is a variation of the Vigenère cipher that works on UTF-8 values for the characters", I considered the UTF-8-character set which consists of 128 characters. Generally, space and delete characters are ignored from the UTF-8-character set making a total of 126 characters, but I have included the space as the 127th character for decrypting the message to a readable plaintext. Then, I computed the below mentioned decryption algorithm [3] by modifying the value used to calculate the mod from 26 to 127. Since I am considering space as a character in the set, I shifted the key value by 1 to accommodate the extra character. Therefore, I assumed and **shifted the key from 'ok' to 'pl'** for implementing this decryption algorithm.

Actual algorithm:
The encryption equation (E) for a Vigenère cipher:
Cipher text, C = E (K, P) = (Pi + Ki) **mod 26**.
The decryption equation (D) for Plain text:
Plain text P = D(C, K) = (Ci -Ki) **mod 26**.
Here Ci, Pi, and Ki are the 'i'th character of cipher text, plain text, and key.

Modified algorithm:
The encryption equation (E) for a Vigenère cipher:
Cipher text, C = E (K, P) = (Pi + Ki) **mod 127**.
The decryption equation (D) for Plain text:
Plain text P = D(C, K) = (Ci -Ki) **mod 127**.
Here Ci, Pi, and Ki are the 'i'th character of cipher text, plain text, and key.

For decryption of the cipher text using the key 'pl', I alternatively coupled the key characters with the ciphertext characters considering 'p' as the first key and 'l' as the next key and so on to perform the modified algorithm.

Firstly, I subtracted the ASCII value of the first letter of the key 'p' from the ASCII value of the first letter of the ciphertext 'E' and computed mod 127 on the result as shown below:
ASCII value of first letter of the ciphertext 'E' is 69 and ASCII value of the first letter of the key 'p' is 112. Using the above decryption algorithm,

(Ci -Ki) mod 127

$(C_1 – K_1)$ mod 127 = (69 – 112) mod 127

= -43 mod 127

= 84 which is **T** when converted to char from ASCII.

Then, I subtracted the ASCII value of the second letter of the key 'l' from the ASCII value of the second letter of the ciphertext 'U' and computed mod 127 on the result as shown below:
ASCII value of second letter of the ciphertext 'U' is 85 and ASCII value of the second letter of the key 'l' is 108. Using the above decryption algorithm,

(Ci -Ki) mod 127

$(C_2 – K_2)$ mod 127 = (85 – 108) mod 127

= -23 mod 127

= 104 which is **h** when converted to char from ASCII.

And so on in a loop for decryption…

Lastly, I converted the resultant ASCII values into printable character and appended the result to the decrypted string.

Therefore, the resulting decrypted string for key 'pl' is:

The~Cape~Barren~goose~(Cereopsis~novaehollandiae[2])~sometimes~also~known~as~the~pig~goose~is~a~species~of~goose~endemic~to~southern~Australia~It~is~a~distinctive~large~grey~bird~that~is~mostly~terrestrial~and~is~not~closely~related~to~other~extant~members~of~the~subfamily~Anserinae

The most challenging part of the problem was to analyse the repeating patterns and writing the code for finding the key and key length. The ciphertext had multiple noticeable repeating patterns which was helpful for frequency analysis. Understanding the Vigenère encryption and fixing the code to map the dictionary for frequency analyses took me about 8 hours of since it was my first time solving an encrypted text.

----------x----------x----------x----------

1.3 Decryption (5 pts)

NnyAagx9[r!Acy4#yh}4^&$(yu%'h&,'nru0^y@Am{w*yg(Ajxu+loyAix45nk%2_&!#ek(My}6ym'#my!#hj41l&"#ly|'m&,+nn4&_t('yl'+hm}0a&+'ak)#no$0&&u0^&*5og!.s&z'_j(A\4&[hv.ct{A`u'Ajru0n&z1ij49cz|Abku&yy*$gk')_jBANny;yiu0yg!5i&x+pk47hjy4qg)'l&z1l&z1ij@Agu''yv'1`ow+_t).s&)*[t41nny4yju$\r}0a&x7]q(Myg#&ysu;yg!5i&(6_g!A`u$&yl'1g&x+po#)yh}4^y45oi|A[y4%iu)

The above is ciphertext, encrypted with a variation of the Vigenère cipher that works on UTF-8 values for the characters. Recover the plaintext.

Solution:

The decrypted plaintext for the above Vigenère ciphertext is:

Thegadwallisabirdofopenwetlands,suchasprairieorsteppelakes,wetgrasslandormarshes withdensefringingvegetation,andusuallyfeedsbydabblingforplantfoodwithheadsubmer ged.Theycanalsodiveunderwaterforfood,moreproficientlythanotherdabblingducks,and mayalsostealfoodfromdivingbirdssuchascoots.

----------x----------x----------x----------

1.4 Analysis (20 pts)

Describe how you recovered the plaintext for the above. What were the challenges? How long did your method take? What features of the cipher made the method easier or harder to execute successfully? Include any code you wrote. You may use comments in your code as part of your description.

Solution:

I used the Kasiski model and frequency analysis methods in my python code to decrypt the given Vigenère ciphertext by following the below steps:

f. Firstly, I analysed the most repetitive trigrams (patterns of 3) from the ciphertext [2] by running a loop I calculated the occurrence of each pattern and added the count to a dictionary. The result of the most repeated patterns of trigrams along with their frequency is:
{2: {'1ij', 'ru0', 'h}4', '(My', 'g!5', 'A`u', '!5i', 'yg!', "yl'", 'l&z', '1l&', '}0a', '0a&', 'yh}', 'z1i', '}4^', '&x+', '.s&', 'u0^', 'Nny', '5i&', 'x+p'}, 3: {'&z1'}}

Here the patterns {'&z1'} is repeated 3 times each, so I considered this pattern as trial and error option to check if this selection is feasible to find the index of occurrence of each of the pattern and calculated the difference in index between each of the occurrences in the ciphertext to understand the key length [1] and splitting the text.

g. To find the indices of the trigram for each of its occurrence, I wrote the second part of the code to run a 'for' loop on the dictionary and appended the result as shown:

{'&z1': [169, 225, 229]}

Here, the trigram '&z1' is occurring at 169th, 225th and 229th index of the cipher text. The difference between these indices gives the length after which the trigram is repeating in the ciphertext as shown below.

{'&z1': [169, 225, 229]}
'&z1': [56, 60, 4]

Next, I am calculating all the factors of these differences in the indices to find the possible key length.

Factors of 4: 1, 2, 4
Factors of 56: 1, 2, 4, 7, 8, 14, 28, 56
Factors of 60: 1, 2, 3, 4, 5, 6, 10, 12, 15, 20, 30, 60

Here, possible key length will be the maximum value of the most repeated factors. Both the factors 2 and 4 are repeated four times as shown above. Since 4 is greater than 2, I considered the possible **key length to be 'four'** based on the Kasiski method of solving Vigenère cipher [1].

h. In step 3, after finding the key length to be '4', I split the entire cipher text by length of 4 for frequency analysis [2]. The result after split will be:

NnyA agx9 [r!A cy4# yh}4 ^&$(yu%' h&,' nru0 ^y@A m{w* yg(A jxu+ loyA ix45 nk%2 _&!# ek(M y}y6 ym'# my!# hj41 l&"# ly|' m&,+ nn4& _t(' yl'+ hm}0 a&+' ak)# no$0 &&u0 ^&*5 og!. s&z' _j(A \ 4& [hv. ct{A `u'A jru0 n&z1 ij49 cz|A bku& yy*$ gk') _jBA Nny; yiu0 yg!5 i&x+ pk47 hjy4 qg)' l&z1 l&z1 ij@A gu'' yv'1 `ow+ _t). s&)* [t41 nny4 yju$ \r}0 a&x7]q(M yg#& ysu; yg!5 i&(6 _g!A `u$& yl'1 g&x+ po#) yh}4 ^y45 oi|A [y4% iu)

i. Next, I computed the frequency analysis of each letter on the split text to find the count of each letter starting at either index 0 or 1 and mapped the count to the respective letters along with their indices. Frequency analysis of split letters for respective index points is shown below:

{'N': {0: 2}, 'n': {1: 4, 0: 6}, 'y': {2: 6, 1: 7, 0: 16}, 'A': {3: 13}, 'a': {0: 4}, 'g': {1: 8, 0: 3}, 'x': {2: 4, 1: 2}, '9': {3: 2}, '[': {0: 4}, 'r': {1: 4}, '!': {2: 7}, 'c': {0: 3}, '4': {2: 10, 3: 4}, '#': {3: 6, 2: 2}, 'h': {1: 3, 0: 4}, '}': {2: 4, 1: 1}, '^': {0: 4}, '&': {1: 17, 3: 5, 0: 1}, '$': {2: 3, 3: 2}, '(': {3: 1, 2: 6}, 'u': {1: 5, 2: 8}, '%': {2: 2, 3: 2}, "'": {3: 8, 2: 7}, ',': {2: 2}, '0': {3: 7}, '@': {2: 2}, 'm': {0: 3, 1: 2}, '{': {1: 1, 2: 1}, 'w': {2: 2}, '*': {3: 2, 2: 2}, 'j': {0: 2, 1: 7}, '+': {3: 6, 2: 1}, 'l': {0: 5, 1: 2}, 'o': {1: 4, 0: 2}, 'i': {0: 6, 1: 2}, '5': {3: 5}, 'k': {1: 6}, '2': {3: 1}, '_': {0: 6}, 'e': {0: 1}, 'M': {3: 2}, '6': {3: 2}, '1': {3: 7}, '"': {2: 1}, '|': {2: 3}, 't': {1: 4}, ')': {2: 5, 3: 2}, '.': {3: 3}, 's': {0: 2, 1: 1}, 'z': {2: 4, 1: 1}, '\\': {0: 2}, ' ': {1: 1}, 'v': {2: 1, 1: 1}, '`': {0: 3}, 'b': {0: 1}, 'B': {2: 1}, ';': {3: 2}, 'p': {0: 2}, '7': {3: 2}, 'q': {0: 1, 1: 1}, ']': {0: 1}}

Here, I observed that letter **'y' is occurring 16 times at 0th index** of the split text, character **'&' is occurring 17 times at the 1st index** of the split text, number **'4' is occurring 10 times at 2nd index** of the split text and letter **'A' is occurring 13 times at 4th index** of the split text making them the possible keys of length 'four' for the encrypted text as described in the Kasiski's model [1]. **Therefore, I consider the key length to be 4 and possible key for decryption to be 'y&4A'**.

j. Now, I computed an algorithm to decrypt the text using the found key 'y&4A'. Since the problem definition mentioned that "the encrypted text is a variation of the Vigenère cipher that works on UTF-8 values for the characters", I considered the UTF-8-character set which consists of 128 characters. Generally, space and delete characters are ignored from the UTF-8-character set making a total of 126 characters, but I have included the space as the 127th character for decrypting the message to a readable plaintext. Then, I computed the below mentioned decryption algorithm [3] by modifying the value used to calculate the mod from 26 to 127.

Now in this scenario, I think the spaces were removed from the plaintext before encrypting it in the problem statement because I considered space as a character in the set just like I considered for the previous problem and shifted the key value by 1 to accommodate the extra character making it 'z'5B' but the shifted key did not work.

So, I considered the key as it is 'y&4A' and the message was decrypted without spaces in them in my code. Therefore, I **considered the key to be 'y&4A'** without making any shifts for implementing this decryption algorithm.

Actual algorithm:
The encryption equation (E) for a Vigenère cipher:
Cipher text, C = E (K, P) = (Pi + Ki) **mod 26**.
The decryption equation (D) for Plain text:
Plain text P = D(C, K) = (Ci -Ki) **mod 26**.
Here Ci, Pi, and Ki are the 'i'th character of cipher text, plain text, and key.

Modified algorithm:
The encryption equation (E) for a Vigenère cipher:
Cipher text, C = E (K, P) = (Pi + Ki) **mod 127**.
The decryption equation (D) for Plain text:
Plain text P = D(C, K) = (Ci -Ki) **mod 127**.
Here Ci, Pi, and Ki are the 'i'th character of cipher text, plain text, and key.

For decryption of the cipher text using the key y&4Al', I alternatively coupled the key characters with the ciphertext characters considering 'y' as the first key and '&' as the second key, '4' as the third key, 'A' as the fourth key and so on to perform the modified algorithm.

Firstly, I subtracted the ASCII value of the first letter of the key 'y' from the ASCII value of the first letter of the ciphertext 'N' and computed mod 127 on the result as shown below:
ASCII value of first letter of the ciphertext 'N' is 78 and ASCII value of the first letter of the key 'y' is 121. Using the above decryption algorithm,

(Ci -Ki) mod 127
$(C_1 – K_1)$ mod 127 = (78 – 121) mod 127
= -43 mod 127
= 84 which is **T** when converted to char from ASCII.

And so on in a loop for decryption…

Lastly, I converted the resultant ASCII values into printable character and appended the result to the decrypted string.

Therefore, the resulting decrypted string for key 'y&4A' is:

Thegadwallisabirdofopenwetlands,suchasprairieorsteppelakes,wetgrasslandormarsheswithdensefringingvegetation,andusuallyfeedsbydabblingforplantfoodwithheadsubmerged.Theycanalsodiveunderwaterforfood,moreproficientlythanotherdabblingducks,andmayalsostealfoodfromdivingbirdssuchascoots.

This problem was not as challenging as the first one since I understood the method for decryption and followed the same process for cracking this message. This ciphertext also had multiple noticeable repeating patterns which was helpful for frequency analysis. This problem took me about 2 hours of since I had solved the first one and just had to minor analysis for solving the encrypted text.

2. SIMPLIFIED DES

2.1 Binary Plaintext (5 pts)

Write the first letter of your first name and first letter of your last name using UTF-8 Basic Latin (ASCII): https://www.unicode.org/charts/PDF/U0000.pdf This is your plaintext (two bytes) for Problem 2.
For example, Thelonious Monk, having initials "'T M" would write: 01010100 01001101**.**

Solution:

My name is **Namana Y Tarikere**. The first letter of my first name is N from Namana and the first letter of my last name is T from Tarikere. The letter N stands for **004E**, and T stands for **0054** in hexadecimal according to the UTF-8 Basic Latin (ASCII) codes list. These hexadecimal numbers are converted to binary where N stands for 01001110 and T stands for 01010100. Therefore, the first letters of my first and last name in binary can be written as **01001110 01010100**.

----------x----------x----------x----------

2.2 Binary Ciphertext (15 pts)

Using Simplified DES, encrypt your plaintext with the following two keys, which differ by one bit:
Key 1: 0110110010
Key 2: 0110010010
By how many bits do the corresponding outputs differ?

Solution:

Consider the result of 2.1 as plaintext for solving this problem.
Plaintext: 01001110 01010100

a. Encrypting the plaintext of my first name letter N with given key 1:

Plaintext for my first name letter N: 01001110
Given Key 1: 0110110010

Input Key 1:

Order	1	2	3	4	5	6	7	8	9	10
Bits	0	1	1	0	1	1	0	0	1	0

Output P10 on Key 1:

Order	3	5	2	7	4	10	1	9	8	6
Bits	1	1	1	0	0	0	0	1	0	1

Dividing P10 keys into 2 halves: **11100 00101**
Left Shift 1 bit on both halves of P10 separately**: 11001 01010**

Input LS-1:

Order	1	2	3	4	5	6	7	8	9	10
Bits	1	1	0	0	1	0	1	0	1	0

Output P8 on LS-1:

Order	6	3	7	4	8	5	10	9
Bits	0	0	1	0	0	1	0	1

Therefore, found 8-bit key 1 is 0010 0101.

Performing Left Shift 2 bits on LS-1 result:
Left Shift - 1: 11001 01010
LS-2 on LS-1: 00111 01001

Input LS-2:

Order	1	2	3	4	5	6	7	8	9	10
Bits	0	0	1	1	1	0	1	0	0	1

Output P8 on LS-2:

Order	6	3	7	4	8	5	10	9
Bits	0	1	1	1	0	1	1	0

Therefore, found 8-bit key 2 is 0111 0110.

Encryption:
Plaintext of N, PT: 0100 1110

Input PT:

Order	1	2	3	4	5	6	7	8
Bits	0	1	0	0	1	1	1	0

Output Initial Permutation (IP) on PT:

Order	2	6	3	1	4	8	5	7
Bits	1	1	0	0	0	0	1	1

Initial Permutation IP: 1100 0011

Input IP:

Order	1	2	3	4	5	6	7	8
Bits	1	1	0	0	0	0	1	1

Output IP^{-1} of IP:

Order	4	1	3	5	7	2	8	6
Bits	0	1	0	0	1	1	1	0

From this, it can be observed that IP inverse of IP is same the plaintext of N.
That is $IP^{-1}(IP) = PT$

Now, to calculate the expanded permutation (EP), consider the initial permutation
IP = 1100 0011
Right part of IP = 0011

Input right part of IP:

Order	1	2	3	4
Bits	0	0	1	1

Output expanded permutation of right part of IP:

Order	4	1	2	3	2	3	4	1
Bits	1	0	0	1	0	1	1	0

Therefore, EP-1 of right part of IP: 1001 0110

Perform ± operation on EP-1 and found key 1:

EP: 1001 0110
Key 1: 0010 0101
± Result: 1011 0011

Get S0 and S1 from the result of ± operation:
First 4 bits of the result is S0 = 1011
Last 4 bits of the result is S1 = 0011

S0 = 1011, row (11) is 3 in decimal and column (01) is 1 in decimal = S0(3, 1)
S1 = 0011, row (01) is 1 in decimal and column (01) is 1 in decimal = S1(1,1)

S0 Table	0	1	2	3
0	1	0	3	2
1	3	2	1	0
2	0	2	1	3
3	3	1	3	2

The value of S0(3,1) from the S0 table is 1 which is 01 in binary.

S1 Table	0	1	2	3
0	0	1	2	3
1	2	0	1	3
2	3	0	1	0
3	2	1	0	3

The value of S1(1,1) from the S1 table is 0 which is 00 in binary.

Therefore, S0S1 = 0100

Input S0S1:

Order	1	2	3	4
Bits	0	1	0	0

Output P4 of S0S1:

Order	2	4	3	1
Bits	1	0	0	0

P4 of S0S1 = 1000
IP = 1100 0011
Now, perform XOR of P4 of S0S1 with left part of IP (1100).

P4 of S0S1: 1000
Left of IP: 1100
± Result: 0100

Result of the above ± XOR function will be swapped with the right part of IP (0011)
Adding XOR result to right part of IP: 0100 0011
After swapping: 0011 0100

Therefore, the new switched text (ST) is 0011 0100.

Now, repeat the entire process with the found 8-bit key 2 (0111 0110)

To find the EP-2, consider right part of the switched text ST (0100)

Input right of ST:

Order	1	2	3	4
Bits	0	1	0	0

Output EP-2 of right part of ST:

Order	4	1	2	3	2	3	4	1
Bits	0	0	1	0	1	0	0	0

Therefore, EP-2 of right part of ST = 0010 1000

Perform ± operation on EP-2 and found key 2:

EP: 0010 1000
Key 2: 0111 0110
± Result: 0101 1110

Get S0 and S1 from the result of ± operation:
First 4 bits of the result is S0 = 0101
Last 4 bits of the result is S1 = 1110

S0 = 0101, row (01) is 1 in decimal and column (10) is 2 in decimal = S0(1,2)
S1 = 1110, row (10) is 2 in decimal and column (11) is 3 in decimal = S1(2,3)

S0 Table	0	1	2	3
0	1	0	3	2
1	3	2	1	0
2	0	2	1	3
3	3	1	3	2

The value of S0(1,2) from the S0 table is 1 which is 01 in binary.

S1 Table	0	1	2	3
0	0	1	2	3
1	2	0	1	3
2	3	0	1	0
3	2	1	0	3

The value of S1(2,3) from the S1 table is 0 which is 00 in binary.

Therefore, S0S1 = 0100

Input S0S1:

Order	1	2	3	4
Bits	0	1	0	0

Output P4 of S0S1:

Order	2	4	3	1
Bits	1	0	0	0

P4 of S0S1 = 1000
Switched text ST = 0011 0100
Now, perform XOR of P4 of S0S1 with left part of ST (0011).

P4 of S0S1: 1000
Left of ST: 0011
± Result: 1011

Result of the above ± XOR function will be swapped with the right part of ST (0100)
Adding XOR result to right part of ST: 1011 0100
Therefore, new IP = 1011 0100

Input new IP:

Order	1	2	3	4	5	6	7	8
Bits	1	0	1	1	0	1	0	0

Output IP^{-1} of IP:

Order	4	1	3	5	7	2	8	6
Bits	1	1	1	0	0	0	0	1

Therefore, the output ciphertext with key 1 for my first name N is 1110 0001.

b. Encrypting the plaintext of my first name letter N with given key 2:

Plaintext for my first name letter N: 01001110
Given Key 2: 01100 10010

Input Key 1:

Order	1	2	3	4	5	6	7	8	9	10
Bits	0	1	1	0	0	1	0	0	1	0

Output P10 on Key 1:

Order	3	5	2	7	4	10	1	9	8	6
Bits	1	0	1	0	0	0	0	1	0	1

Dividing P10 keys into 2 halves: **10100 00101**
Left Shift 1 bit on both halves of P10 separately**: 01001 01010**

Input LS-1:

Order	1	2	3	4	5	6	7	8	9	10
Bits	0	1	0	0	1	0	1	0	1	0

Output P8 on LS-1:

Order	6	3	7	4	8	5	10	9
Bits	0	0	1	0	0	1	0	1

Therefore, found 8-bit key 1 is 0010 0101.

Performing Left Shift 2 bits on LS-1 result:
Left Shift - 1: 01001 01010
LS-2 on LS-1: 00101 01001

Input LS-2:

Order	1	2	3	4	5	6	7	8	9	10
Bits	0	0	1	0	1	0	1	0	0	1

Output P8 on LS-2:

Order	6	3	7	4	8	5	10	9
Bits	0	1	1	0	0	1	1	0

Therefore, found 8-bit key 2 is 0110 0110.

Encryption:
Plaintext of N, PT: 0100 1110

Input PT:

Order	1	2	3	4	5	6	7	8
Bits	0	1	0	0	1	1	1	0

Output Initial Permutation (IP) on PT:

Order	2	6	3	1	4	8	5	7
Bits	1	1	0	0	0	0	1	1

Initial Permutation IP: 1100 0011

Input IP:

Order	1	2	3	4	5	6	7	8
Bits	1	1	0	0	0	0	1	1

Output IP^{-1} of IP:

Order	4	1	3	5	7	2	8	6
Bits	0	1	0	0	1	1	1	0

From this, it can be observed that IP inverse of IP is same the plaintext of N.
That is IP^{-1}(IP) = PT

Now, to calculate the expanded permutation (EP), consider the initial permutation
IP = 1100 0011
Right part of IP = 0011

Input right part of IP:

Order	1	2	3	4
Bits	0	0	1	1

Output expanded permutation of right part of IP:

Order	4	1	2	3	2	3	4	1
Bits	1	0	0	1	0	1	1	0

Therefore, EP-1 of right part of IP: 1001 0110

Perform ± operation on EP-1 and found key 1:

EP: 1001 0110
Key 1: 0010 0101
± Result: 1011 0011

Get S0 and S1 from the result of ± operation:
First 4 bits of the result is S0 = 1011
Last 4 bits of the result is S1 = 0011

S0 = 1011, row (11) is 3 in decimal and column (01) is 1 in decimal = S0(3, 1)
S1 = 0011, row (01) is 1 in decimal and column (01) is 1 in decimal = S1(1,1)

S0 Table	0	1	2	3
0	1	0	3	2
1	3	2	1	0
2	0	2	1	3
3	3	1	3	2

The value of S0(3,1) from the S0 table is 1 which is 01 in binary.

S1 Table	0	1	2	3
0	0	1	2	3
1	2	0	1	3
2	3	0	1	0
3	2	1	0	3

The value of S1(1,1) from the S1 table is 0 which is 00 in binary.

Therefore, S0S1 = 0100

Input S0S1:

Order	1	2	3	4
Bits	0	1	0	0

Output P4 of S0S1:

Order	2	4	3	1
Bits	1	0	0	0

P4 of S0S1 = 1000
IP = 1100 0011
Now, perform XOR of P4 of S0S1 with left part of IP (1100).

P4 of S0S1: 1000
Left of IP: 1100
± Result: 0100

Result of the above ± XOR function will be swapped with the right part of IP (0011)
Adding XOR result to right part of IP: 0100 0011
After swapping: 0011 0100

Therefore, the new switched text (ST) is 0011 0100.

Now, repeat the entire process with the found 8-bit key 2 (0110 0110)

To find the EP-2, consider right part of the switched text ST (0100)

Input right of ST:

Order	1	2	3	4
Bits	0	1	0	0

Output EP-2 of right part of ST:

Order	4	1	2	3	2	3	4	1
Bits	0	0	1	0	1	0	0	0

Therefore, EP-2 of right part of ST = 0010 1000

Perform ± operation on EP-2 and found key 2:

EP: 0010 1000
Key 2: 0110 0110
± Result: 0100 1110

Get S0 and S1 from the result of ± operation:
First 4 bits of the result is S0 = 0100
Last 4 bits of the result is S1 = 1110

S0 = 0100, row (00) is 0 in decimal and column (10) is 2 in decimal = S0(0,2)
S1 = 1110, row (10) is 2 in decimal and column (11) is 3 in decimal = S1(2,3)

S0 Table	0	1	2	3
0	1	0	3	2
1	3	2	1	0
2	0	2	1	3
3	3	1	3	2

The value of S0(0,2) from the S0 table is 3 which is 11 in binary.

S1 Table	0	1	2	3
0	0	1	2	3
1	2	0	1	3
2	3	0	1	0
3	2	1	0	3

The value of S1(2,3) from the S1 table is 0 which is 00 in binary.

Therefore, S0S1 = 1100

Input S0S1:

Order	1	2	3	4
Bits	1	1	0	0

Output P4 of S0S1:

Order	2	4	3	1
Bits	1	0	0	1

P4 of S0S1 = 1001
Switched text ST = 0011 0100
Now, perform XOR of P4 of S0S1 with left part of ST (0011).

P4 of S0S1: 1001
Left of ST: 0011
± Result: 1010

Result of the above ± XOR function will be swapped with the right part of ST (0100)
Adding XOR result to right part of ST: 1010 0100
Therefore, new IP = 1010 0100

Input new IP:

Order	1	2	3	4	5	6	7	8
Bits	1	0	1	0	0	1	0	0

Output IP^{-1} of IP:

Order	4	1	3	5	7	2	8	6
Bits	0	1	1	0	0	0	0	1

Therefore, the output ciphertext with key 2 for my first name N is 0110 0001.

c. **Encrypting the plaintext of my last name letter T with given key 1:**

Plaintext for my last name letter T: 01010100
Given Key 1: 01101 10010

Input Key 1:

Order	1	2	3	4	5	6	7	8	9	10
Bits	0	1	1	0	1	1	0	0	1	0

Output P10 on Key 1:

Order	3	5	2	7	4	10	1	9	8	6
Bits	1	1	1	0	0	0	0	1	0	1

Dividing P10 keys into 2 halves: **11100 00101**
Left Shift 1 bit on both halves of P10 separately**: 11001 01010**

Input LS-1:

Order	1	2	3	4	5	6	7	8	9	10
Bits	1	1	0	0	1	0	1	0	1	0

Output P8 on LS-1:

Order	6	3	7	4	8	5	10	9
Bits	0	0	1	0	0	1	0	1

Therefore, found 8-bit key 1 is 0010 0101.

Performing Left Shift 2 bits on LS-1 result:
Left Shift - 1: 11001 01010
LS-2 on LS-1: 00111 01001

Input LS-2:

Order	1	2	3	4	5	6	7	8	9	10
Bits	0	0	1	1	1	0	1	0	0	1

Output P8 on LS-2:

Order	6	3	7	4	8	5	10	9
Bits	0	1	1	1	0	1	1	0

Therefore, found 8-bit key 2 is 0111 0110.

Encryption:
Plaintext of T, PT: 0101 0100

Input PT:

Order	1	2	3	4	5	6	7	8
Bits	0	1	0	1	0	1	0	0

Output Initial Permutation (IP) on PT:

Order	2	6	3	1	4	8	5	7
Bits	1	1	0	0	1	0	0	0

Initial Permutation IP: 1100 1000

Input IP:

Order	1	2	3	4	5	6	7	8
Bits	1	1	0	0	1	0	0	0

Output IP^{-1} of IP:

Order	4	1	3	5	7	2	8	6
Bits	0	1	0	1	0	1	0	0

From this, it can be observed that IP inverse of IP is same the plaintext of T.
That is IP^{-1}(IP) = PT

Now, to calculate the expanded permutation (EP), consider the initial permutation
IP = 1100 1000
Right part of IP = 1000

Input right part of IP:

Order	1	2	3	4
Bits	1	0	0	0

Output expanded permutation of right part of IP:

Order	4	1	2	3	2	3	4	1
Bits	0	1	0	0	0	0	0	1

Therefore, EP-1 of right part of IP: 0100 0001

Perform ± operation on EP-1 and found key 1:

EP: 0100 0001
Key 1: 0010 0101
± Result: 0110 0100

Get S0 and S1 from the result of ± operation:
First 4 bits of the result is S0 = 0110
Last 4 bits of the result is S1 = 0100

S0 = 0110, row (00) is 0 in decimal and column (11) is 3 in decimal = S0(0,3)
S1 = 0100, row (00) is 0 in decimal and column (10) is 2 in decimal = S1(0,2)

S0 Table	0	1	2	3
0	1	0	3	2
1	3	2	1	0
2	0	2	1	3
3	3	1	3	2

The value of S0(0,3) from the S0 table is 2 which is 10 in binary.

S1 Table	0	1	2	3
0	0	1	2	3
1	2	0	1	3
2	3	0	1	0
3	2	1	0	3

The value of S1(0,2) from the S1 table is 2 which is 10 in binary.

Therefore, S0S1 = 1010

Input S0S1:

Order	1	2	3	4
Bits	1	0	1	0

Output P4 of S0S1:

Order	2	4	3	1
Bits	0	0	1	1

P4 of S0S1 = 0011
IP = 1100 1000
Now, perform XOR of P4 of S0S1 with left part of IP (1100).

P4 of S0S1: 0011
Left of IP: 1100
± Result: 1111

Result of the above ± XOR function will be swapped with the right part of IP (1000)
Adding XOR result to right part of IP: 1111 1000
After swapping: 1000 1111

Therefore, the new switched text (ST) is 1000 1111.

Now, repeat the entire process with the found 8-bit key 2 (0111 0110)

To find the EP-2, consider right part of the switched text ST (1111)

Input right of ST:

Order	1	2	3	4
Bits	1	1	1	1

Output EP-2 of right part of ST:

Order	4	1	2	3	2	3	4	1
Bits	1	1	1	1	1	1	1	1

Therefore, EP-2 of right part of ST = 1111 1111

Perform ± operation on EP-2 and found key 2:

EP: 1111 1111
Key 2: 0111 0110
± Result: 1000 1001

Get S0 and S1 from the result of ± operation:
First 4 bits of the result is S0 = 1000
Last 4 bits of the result is S1 = 1001

S0 = 1000, row (10) is 2 in decimal and column (00) is 0 in decimal = S0(2,0)
S1 = 1001, row (11) is 3 in decimal and column (00) is 0 in decimal = S1(3,0)

S0 Table	0	1	2	3
0	1	0	3	2
1	3	2	1	0
2	0	2	1	3
3	3	1	3	2

The value of S0(2,0) from the S0 table is 0 which is 00 in binary.

S1 Table	0	1	2	3
0	0	1	2	3
1	2	0	1	3
2	3	0	1	0
3	2	1	0	3

The value of S1(3,0) from the S1 table is 2 which is 10 in binary.

Therefore, S0S1 = 0010

Input S0S1:

Order	1	2	3	4
Bits	0	0	1	0

Output P4 of S0S1:

Order	2	4	3	1
Bits	0	0	1	0

P4 of S0S1 = 0010
Switched text ST = 1000 1111
Now, perform XOR of P4 of S0S1 with left part of ST (1000).

P4 of S0S1: 0010
Left of ST: 1000
± Result: 1010

Result of the above ± XOR function will be swapped with the right part of ST (1111)

Adding XOR result to right part of ST: 1010 1111
Therefore, new IP = 1010 1111

Input new IP:

Order	1	2	3	4	5	6	7	8
Bits	1	0	1	0	1	1	1	1

Output IP^{-1} of IP:

Order	4	1	3	5	7	2	8	6
Bits	0	1	1	1	1	0	1	1

Therefore, the output ciphertext with key 1 for my last name T is 0111 1011.

d. Encrypting the plaintext of my last name letter T with given key 2:

Plaintext for my last name letter T: 01010100
Given Key 2: 01100 10010

Input Key 1:

Order	1	2	3	4	5	6	7	8	9	10
Bits	0	1	1	0	0	1	0	0	1	0

Output P10 on Key 1:

Order	3	5	2	7	4	10	1	9	8	6
Bits	1	0	1	0	0	0	0	1	0	1

Dividing P10 keys into 2 halves: **10100 00101**
Left Shift 1 bit on both halves of P10 separately**: 01001 01010**

Input LS-1:

Order	1	2	3	4	5	6	7	8	9	10
Bits	0	1	0	0	1	0	1	0	1	0

Output P8 on LS-1:

Order	6	3	7	4	8	5	10	9
Bits	0	0	1	0	0	1	0	1

Therefore, found 8-bit key 1 is 0010 0101.

Performing Left Shift 2 bits on LS-1 result:
Left Shift - 1: 01001 01010
LS-2 on LS-1: 00101 01001

Input LS-2:

Order	1	2	3	4	5	6	7	8	9	10
Bits	0	0	1	0	1	0	1	0	0	1

Output P8 on LS-2:

Order	6	3	7	4	8	5	10	9
Bits	0	1	1	0	0	1	1	0

Therefore, found 8-bit key 2 is 0110 0110.

Encryption:
Plaintext of T, PT: 0101 0100

Input PT:

Order	1	2	3	4	5	6	7	8
Bits	0	1	0	1	0	1	0	0

Output Initial Permutation (IP) on PT:

Order	2	6	3	1	4	8	5	7
Bits	1	1	0	0	1	0	0	0

Initial Permutation IP: 1100 1000

Input IP:

Order	1	2	3	4	5	6	7	8
Bits	1	1	0	0	1	0	0	0

Output IP^{-1} of IP:

Order	4	1	3	5	7	2	8	6
Bits	0	1	0	1	0	1	0	0

From this, it can be observed that IP inverse of IP is same the plaintext of T.
That is IP^{-1}(IP) = PT

Now, to calculate the expanded permutation (EP), consider the initial permutation
IP = 1100 1000
Right part of IP = 1000

Input right part of IP:

Order	1	2	3	4
Bits	1	0	0	0

Output expanded permutation of right part of IP:

Order	4	1	2	3	2	3	4	1
Bits	0	1	0	0	0	0	0	1

Therefore, EP-1 of right part of IP: 0100 0001

Perform ± operation on EP-1 and found key 1:

EP: 0100 0001
Key 1: 0010 0101
± Result: 0110 0100

Get S0 and S1 from the result of ± operation:
First 4 bits of the result is S0 = 0110
Last 4 bits of the result is S1 = 0100

S0 = 0110, row (00) is 0 in decimal and column (11) is 3 in decimal = S0(0,3)
S1 = 0100, row (00) is 0 in decimal and column (10) is 2 in decimal = S1(0,2)

S0 Table	0	1	2	3
0	1	0	3	2
1	3	2	1	0
2	0	2	1	3
3	3	1	3	2

The value of S0(0,3) from the S0 table is 2 which is 10 in binary.

S1 Table	0	1	2	3
0	0	1	2	3
1	2	0	1	3
2	3	0	1	0
3	2	1	0	3

The value of S1(0,2) from the S1 table is 2 which is 10 in binary.

Therefore, S0S1 = 1010

Input S0S1:

Order	1	2	3	4
Bits	1	0	1	0

Output P4 of S0S1:

Order	2	4	3	1
Bits	0	0	1	1

P4 of S0S1 = 0011
IP = 1100 1000
Now, perform XOR of P4 of S0S1 with left part of IP (1100).

```
    P4 of S0S1: 0011
  Left of IP: 1100
   ± Result: 1111
```

Result of the above ± XOR function will be swapped with the right part of IP (1000)
Adding XOR result to right part of IP: 1111 1000
After swapping: 1000 1111

Therefore, the new switched text (ST) is 1000 1111.

Now, repeat the entire process with the found 8-bit key 2 (0110 0110)

To find the EP-2, consider right part of the switched text ST (1111)

Input right of ST:

Order	1	2	3	4
Bits	1	1	1	1

Output EP-2 of right part of ST:

Order	4	1	2	3	2	3	4	1
Bits	1	1	1	1	1	1	1	1

Therefore, EP-2 of right part of ST = 1111 1111

Perform ± operation on EP-2 and found key 2:

EP: 1111 1111
Key 2: 0110 0110
± Result: 1001 1001

Get S0 and S1 from the result of ± operation:
First 4 bits of the result is S0 = 1001
Last 4 bits of the result is S1 = 1001

S0 = 1001, row (11) is 3 in decimal and column (00) is 0 in decimal = S0(3,0)
S1 = 1001, row (11) is 3 in decimal and column (00) is 0 in decimal = S1(3,0)

S0 Table	0	1	2	3
0	1	0	3	2
1	3	2	1	0
2	0	2	1	3
3	3	1	3	2

The value of S0(3,0) from the S0 table is 3 which is 11 in binary.

S1 Table	0	1	2	3
0	0	1	2	3
1	2	0	1	3
2	3	0	1	0
3	2	1	0	3

The value of S1(3,0) from the S1 table is 2 which is 10 in binary.

Therefore, S0S1 = 1110
Input S0S1:

Order	1	2	3	4
Bits	1	1	1	0

Output P4 of S0S1:

Order	2	4	3	1
Bits	1	0	1	1

P4 of S0S1 = 1011
Switched text ST = 1000 1111
Now, perform XOR of P4 of S0S1 with left part of ST (1000).

P4 of S0S1: 1011
Left of ST: 1000
± Result: 0011

Result of the above ± XOR function will be swapped with the right part of ST (1111)
Adding XOR result to right part of ST: 0011 1111
Therefore, new IP = 0011 1111

Input new IP:

Order	1	2	3	4	5	6	7	8
Bits	0	0	1	1	1	1	1	1

Output IP^{-1} of IP:

Order	4	1	3	5	7	2	8	6
Bits	1	0	1	1	1	0	1	1

Therefore, the output ciphertext with key 2 for my last name T is 1011 1011.

Details	Plaintext	Ciphertext	Key
Key 1 with my first name N	01001110	11100001	0110110010
Key 2 with my first name N	01001110	01100001	0110010010
Key 1 with my last name T	01010100	01111011	0110110010
Key 2 with my last name T	01010100	10111011	0110010010

For the first case, for my first name N, the corresponding ciphertext output differs by one bit for key 1 and key 2 as shown in green colour in the above table with respect to their positions. For the second case, for my last name T, the corresponding ciphertext output differs by 2 bits for key 1 and key 2 as shown in pink colour in the above table with respect to their positions. The overall output differs by 3 bits for the ciphertexts of first name and last name by using key 1 and key 2 irrespective of their position.

Plaintext of first letter of my first name N and last name T:

01001110 01010100

Ciphertext of the plaintext with key 1:

11100001 01111011

Ciphertext of the plaintext with key 2:

01100001 10111011

----------x----------x----------x----------

2.3 Attacking Simplified DES (5 pts)

Consider the following SDES plaintext-ciphertext pairs:

Plaintext 1: 01110110 01101111 01110100 01100101 01110010 01110011
Ciphertext 1: 10101110 01110010 10001001 01010111 00101111 01111010

Plaintext 2: 01100001 01101110 01101001 01101101 01100001 01101100 01110011
Ciphertext 2: 00100100 10000000 10011010 11111111 00100100 00011000 11111100

Plaintext 3: 01110011 01100001 01101100 01100001 01100100
Ciphertext 3: 01111010 11010010 01001110 11010010 11110101

Two of the pairs were encrypted with the same key. Which two? Discuss the method you used to determine the result and compare the process of recovering the key from a plaintext-ciphertext pair to brute-force determination of an SDES key.

Solution:

When the given pairs of plaintexts and their respective ciphertexts are closely analysed, it can be found that the plaintext **01110011** is common in 3 sets and their respective ciphertext (01111010) are same for pairs 1 and 3 as indicated in green colour above. But the ciphertext (11111100) of the same plaintext does not match with the ciphertext of either of the pairs 1 and 3 as indicated in green and blue colour above. We know that identical ciphertexts can be formed only when both the plaintext and keys are same, hence the keys of pair 1 and pair 3 are the same. Additionally, the plaintext **01100001** is common in pairs 2 and 3 but their ciphertexts do not match as shown in red colour above. The plaintext **01101100** is also common between pairs 2 and 3 but their respective ciphertexts do not match as shown in pink colour above. Hence, pairs 2 and 3 are encrypted with different keys since they have different cipher texts. **Therefore, it can be concluded that the keys for encrypting the plaintext of pair 1 and pair 3 are same.**
One of the most common methods for decrypting the ciphertext in SDES encryption is a Brute-force attack, which includes all combinations of key sets. The brute-force attack uses a 10-bit key making it too easy to attack since the standard DES algorithm uses a 64-bit key which is stronger and complex than Simplified DES.

The method of finding the key using the given plaintext and ciphertext pair is easy compared to Brute-force attack because we can follow the SDES encryption method to find the key for the pair of plaintexts and ciphertext that we already have. In this problem for example, we can easily identify repeating patterns of same plaintext in all three pairs and compare their respective ciphertexts for the exact matching values to find the pairs which uses the same key for encryption.

Therefore, obtaining the key from analysing pairs of plaintexts and ciphertext is more effective compared to brute-force attack.

---

3. CRYPTOGRAPHIC METHODS

Alice wishes to transmit 15 TB of data to Bob by mailing him a hard drive. She would like the information to remain confidential, such that only she and Bob can read the information. Consider the following methods:

• AES-256 • SHA-2 • PKCS #7 • DES • One-Time Pad • Pseudorandom Numbers • RSA

Select two of these methods and comment (in no more than 400 words) as to the suitability of each method for Alice's application. Choose one method that is appropriate and one method that is not. If the appropriate method requires additional steps beyond performing a cryptographic operation on the data and mailing the hard drive, describe those steps. For the inappropriate method, describe why it is inappropriate.

Solution:

AES-256 encryption is the best method for Alice to send data confidentially to Bob. AES (Advanced Encryption Standard) is a symmetric encryption algorithm which is more efficient than DES. AES provides security in both hardware and software level to efficiently encrypt huge amount of data. After encrypting the data, Alice should share a secret key to Bob to decrypt the information using a secure file transfer or encrypted mail. AES algorithm uses multiple rounds of keys derived from the main key for encryption and higher the block size, higher is the number of rounds of encryption on the data.

The other methods have various issues to encrypt 15TB of data.

SHA-2 is a Secure Hashing algorithm used for authenticating data and password hashing which is outdated. SHA-2 provides security to the software and file integrity than confidentiality making it unsuitable for encrypting passwords and harddrives [4].

PKCS #7 (Public Key Cryptography Standard #7) is used for encrypting and enveloping small amount of data and does not support secure key exchange between Alice and Bob making it unsuitable for this scenario [5].

One-Time Pad method uses a key of length which is greater than or equal to length of plaintext (Key Length ≥ Plain text), making it not suitable for generating key, encryption, and transmission of 15TB of data.

Pseudorandom Numbers use deterministic algorithms that lack true randomness required for strong encryption of data and these patterns are easily predictable and repeatable making it unsuitable for Alice's application.

DES is still in use even though it is outdated and not recommended to use since has a smaller key size and more vulnerable to attacks. AES is more efficient and provides more security since is uses a 256-bit size key than DES.

RSA is an asymmetric encryption algorithm which is very efficient for data encryption and transmission between two people but encrypting and decrypting 15TB of data using this technique is not practical as it is too time consuming and slower than AES.

Therefore, for Alice's application, **DES is the worst method of encryption and AES with 256-bit key is the best suitable method**, since it has the best security features, never been solved before, and works faster for large data. Secure key exchange can be achieved by creating a secure communication channel using asymmetric RSA encryption technique for sharing the keys between Bob and Alice along with AES-256 algorithm.

References:

[1] Article on Kasiski method, Available: Cryptography -- Vigenere Cipher (uu.se)

[2] M. VandeWettering, "*How can I crack the Vigenère cipher without knowing the key*?", Available: Mark VandeWettering's answer to How can I crack the Vigenere cipher without knowing the key? - Quora

[3] C. Partha, "*The original 26*26 Vigenère table*", Available: The Original 26 × 26 Vigenere table [13]. The encryption equation (E)... | Download Scientific Diagram (researchgate.net)

[4] J. Lake*, "What is SHA-2 and how does it work? Guide to cryptographic hash functions"*, 17th February 2022, Available: https://www.comparitech.com/blog/information-security/what-is-sha-2-how-does-it-work/

[5] A. Alvin, et.al, *"PKCS #7 Cryptographic Messaging Syntax Concepts"*, 7th January 2021, Available: https://learn.microsoft.com/en-us/windows/win32/seccrypto/pkcs--7-concepts