Cryptography Project Report

Introduction:

This project designs and implements a fully functional symmetric encryption-decryption algorithm pair in C++. The user is able to generate a random encryption key to encrypt and decrypt a text of any given length.

Compiling and running instructions:

- The program needs to be compiled with C++ 11 with all its classes and the main. (Command: g++ -std=c++11 main.cpp Encrypter.hpp Encrypter.cpp Decrypter.hpp Decrypter.cpp)
- By default, the program will look for a "*message.txt*" file in the same folder for encryption purposes.
  There is also an option to pass the file name as a Command Line Argument. This code is commented out by default but can be used too.
- Once the program is executed, it will display a 128-bit key and the time it took for encryption in milliseconds.
- The program prompts the user to enter the destination where the encrypted text is stored. A *'.txt'* file needs to be entered. The program does not continue until the input is valid. By default, the program creates a file named '*encrypted.txt*' with the encrypted content.
- Once the filename is entered and accepted, the program prompts the user for the decryption key.
- The same key as provided during encryption should be used in order to decrypt the message.
- The decrypted message is displayed on the console with the decryption time in milliseconds.

Documentation:

Detailed documentation of the program can be found in the folder named "documentation". The documentation is generated with Doxygen and it includes documentation in HTML, latex and RTF format.

To use the HTML documentation, simply open the "index.html" file in the "html" folder under "documentation". Latex users can use CMAKE to generate documentation from the "latex" folder. A simple RTF documentation is also included in the "rtf" folder.

Description of Algorithm:

The encryption algorithm generates and uses a randomly generated 128-bit key. This key is presented to the user before encryption and the program generates a text file with the encrypted text. The program with the same key is then used to decrypt the encrypted text to get back the original plain text. The program works as follows:

1. Input reading and Key generation:

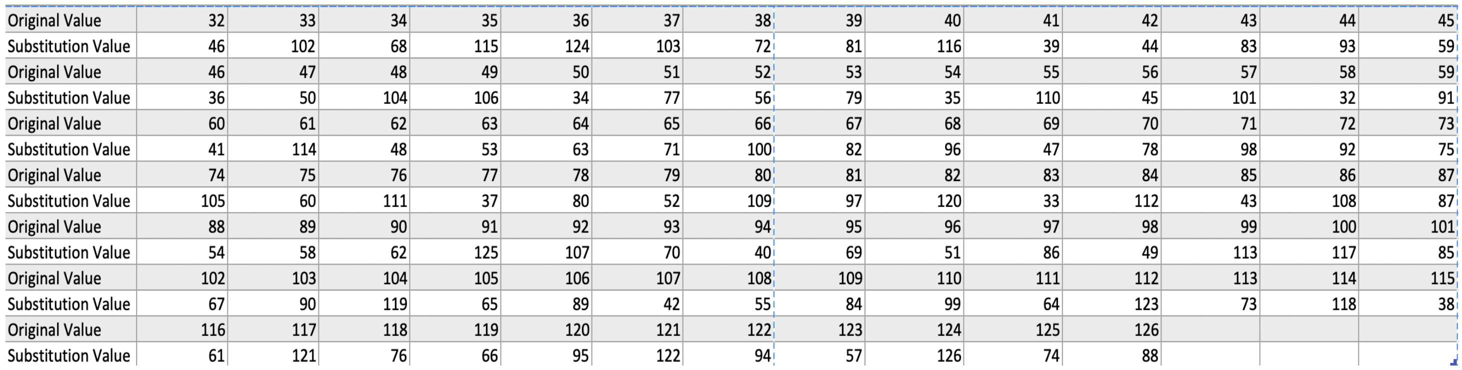
   Initially, the program reads an input text for encryption purposes from a text file and generates a random 128-bit key. The key is 16 characters long and contains alpha-numeric values. These values are generated randomly by using the random function from the C Standard General Utilities Library. The seed is picked as the current system time so that the sequence of random numbers is different for each use case. The key is parsed and used in different parts of the encryption algorithm. The eleventh position of the key is always a numeric value and is used for Caesar cipher encryption in the algorithm. Further details can be found in Section 7: Caesar’s cipher encryption.

2. Vigenère encryption (1):

   The input text goes through Vigenère encryption as the first encryption step. The encryption algorithm is custom written by considering the printable ASCII characters which ranges in decimal value from 32 to 126. The control and extended ASCII table is not considered because it brings complications when the text is written and read from a file as they are not standard characters.
   The first Vigenère encryption uses the last 5 bytes, which is the last 5 characters of the key. The input text is split into 5 characters each by using the substring method. The ASCII value of each 5 characters is added with the ASCII value of the 5-character key, resulting in an encrypted text using the Vigenère method of encryption. To keep the values in the range of printable ASCII characters, the program checks if the resulting value goes over 126. In such condition, the value is recalculated as $value = (value - 127) + 32$. This keeps the result value range in between 32 and 126.
   The resulting encrypted text is then passed to a custom substitution box.

3. Custom substitution box (S-box):

   A custom substitution box is generated by using a short program that randomly shuffles numbers in the range from 32 to 126. This program can be found under “Trash methods” inside the file “Encrypter.hpp”. The substitution box is declared as a public array in the Encrypter class. The ASCII values of each character from the text are substituted according to Table 1.0.



Original Value	32	33	34	35	36	37	38	39	40	41	42	43	44	45
Substitution Value	46	102	68	115	124	103	72	81	116	39	44	83	93	59
Original Value	46	47	48	49	50	51	52	53	54	55	56	57	58	59
Substitution Value	36	50	104	106	34	77	56	79	35	110	45	101	32	91
Original Value	60	61	62	63	64	65	66	67	68	69	70	71	72	73
Substitution Value	41	114	48	53	63	71	100	82	96	47	78	98	92	75
Original Value	74	75	76	77	78	79	80	81	82	83	84	85	86	87
Substitution Value	105	60	111	37	80	52	109	97	120	33	112	43	108	87
Original Value	88	89	90	91	92	93	94	95	96	97	98	99	100	101
Substitution Value	54	58	62	125	107	70	40	69	51	86	49	113	117	85
Original Value	102	103	104	105	106	107	108	109	110	111	112	113	114	115
Substitution Value	67	90	119	65	89	42	55	84	99	64	123	73	118	38
Original Value	116	117	118	119	120	121	122	123	124	125	126			
Substitution Value	61	121	76	66	95	122	94	57	126	74	88			

Table 1.0: Custom Substitution Box (S-box)

4. <u>Vigenère encryption (2):</u>

   The resulting text goes through another series of Vigenère style encryption. On this step, the first 5 bytes are used for encryption.

5. <u>Custom Permutation box (P-box):</u>

   The encrypted text is then passed through a custom P-box. The positions of each five characters in the text are permuted as shown in Table 2.0.

Initial Position	0	1	2	3	4
Permuted Position	2	4	1	0	3

Table 2.0: Custom Permutation Box (P-box)

Since permutation is done for each five characters, the program checks the length of the text to ensure that the text can be permuted. If the length of the text is not a multiple of 5, the program appends n pipe characters ('|'), where $n = 5 - ((length\ of\ text)\ \%\ 5)$.

6. <u>Vigenère encryption (3):</u>

   The encrypted text is further passed through a final series of Vigenère encryption. Here, bytes 6-10 is used for encryption.

7. <u>Caesar's cipher encryption:</u>

   After the final Vigenère encryption, the text goes through Caesar's cipher encryption. This encryption step uses the numeric value on the eleventh position of the key. The value is added to the ASCII value of each character in the text. A range check similar to the one done in Vigenère encryption is present to ensure that the values lie in the printable ASCII character range.

8. <u>5 block Feistel style encryption:</u>

On the final encryption step, a 5-block Feistel style encryption is implemented. On this step, both the input text and key are converted into binary and stored in a vector of bitsets. Each bitset is 8-bit long. As a result, sixteen 8-bit sub-keys are generated.
The input text is divided into groups of 5 bitsets each. Since the text received is a multiple of 5 as a result of the P-box permutation, the grouping is uniform. Next, each group is transformed using a different sub-key for each round for a total of 16 rounds. The transformation is shown in Figure 1.0 and Figure 1.1.

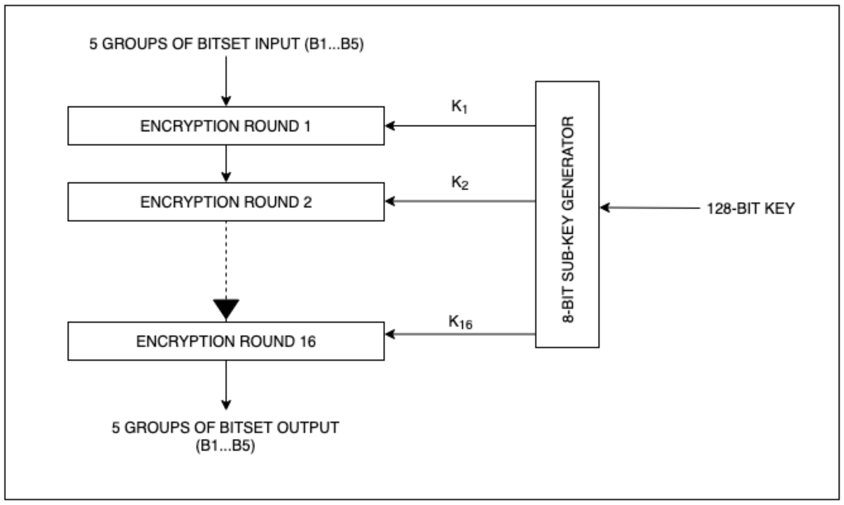


Figure 1.0: Encryption rounds using different sub-keys.

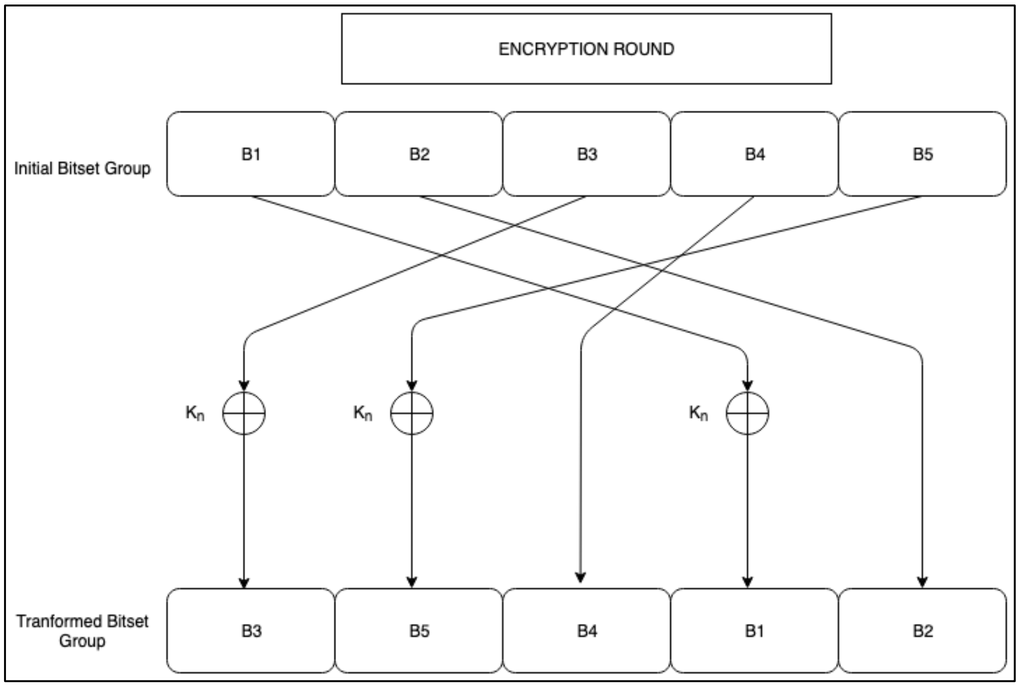


Figure 1.1: Transformation in each Encryption round.

Reversibility and Decryption:

The algorithm is fully reversible and the input plaintext can be decrypted by using the same key. The steps to encrypt are simultaneously done in reverse to decrypt the cipher text. The reverse of the 5 Block Feistel encryption is shown in Figure 2.0 and 2.1. As it is a Feistel structure encryption, the key schedule is reversed to decrypt the cipher text [1].

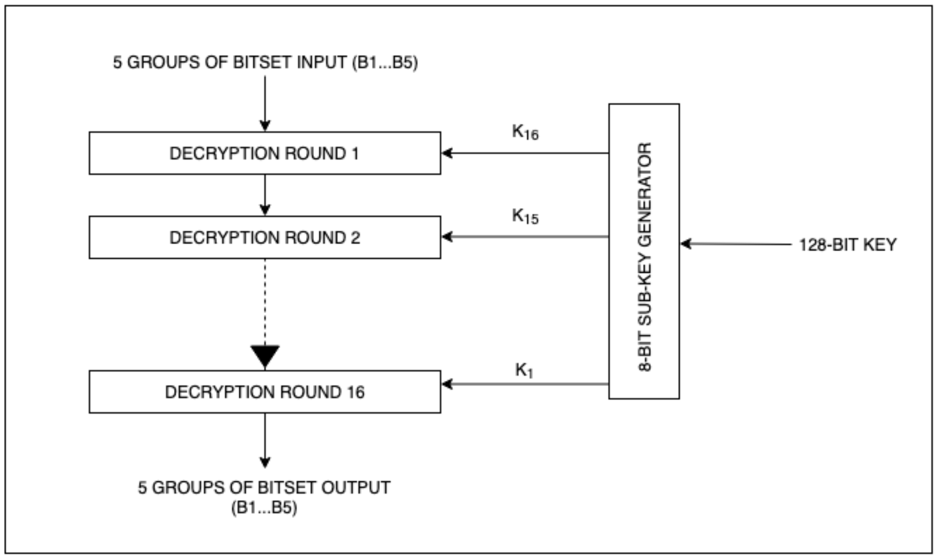


Figure 2.0: Decryption rounds using different sub-keys.

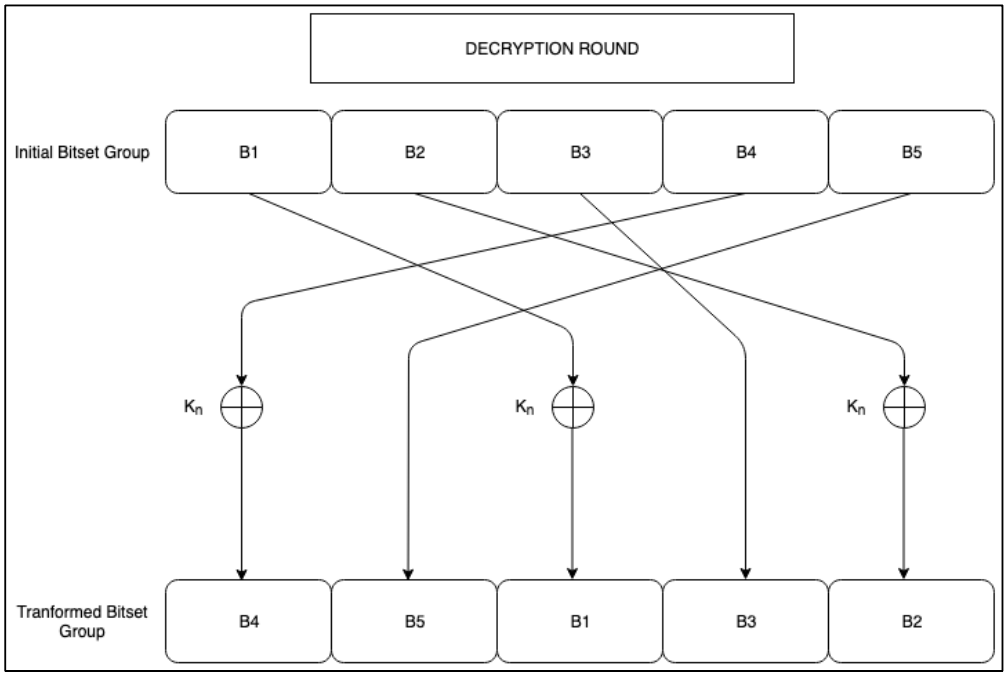


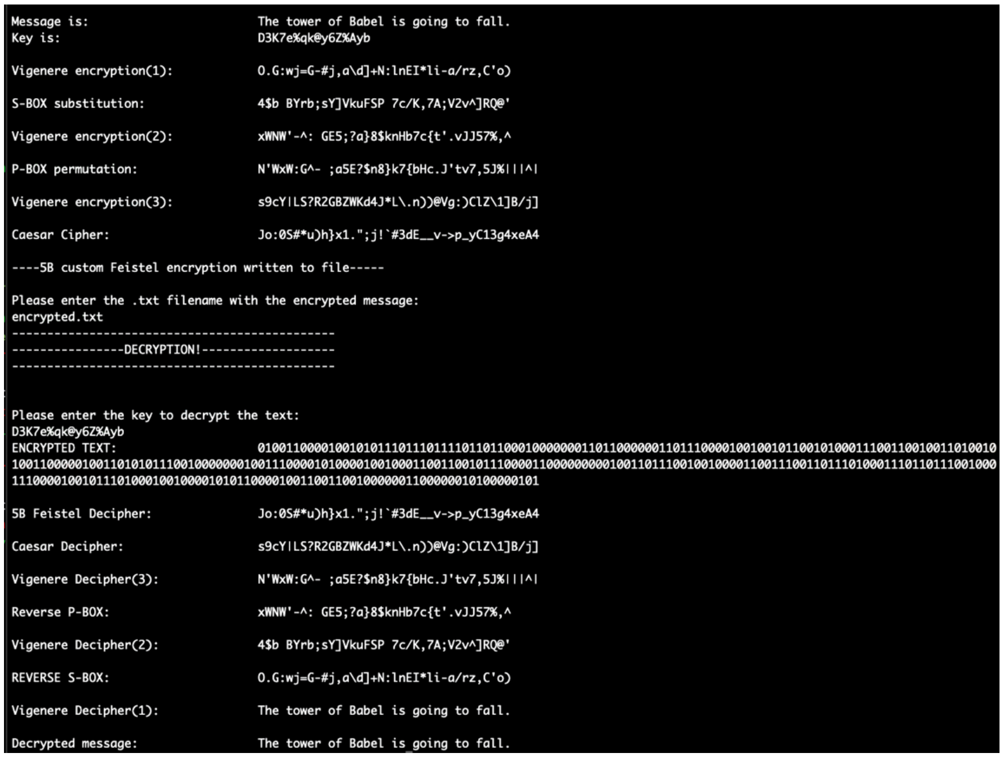
Figure 2.1: Transformation in each Decryption round.

In both Caesar and Vigènere cipher decryption, the value of the same sub-key used while encrypting is subtracted from the cipher text to get the decrypted plain text. The reverse of the P-box is shown in Table 3.0, which is used to reverse the permutation to decrypt the cipher text.

Initial Position	0	1	2	3	4
Permuted Position	3	2	0	4	1

Table 3.0: Custom Reverse Permutation Box (Reverse P-box)

The S-box is stored as a public member array in the Encrypter class. In order to reverse the S-box substitution and get the original values to decrypt the cipher text, the program needs to iterate through the array to find the substituted value. When found, the index number gives the original value. The use of this data structure simulates a behavior similar to a one-way-trap function [1] as decryption has a linear time complexity of *O(n)* while encryption has a constant time complexity of *O(1)*. This is because we can directly get the substitution value by accessing the respective index number during encryption but not the other way around.
In this way, the algorithm does the reverse of each encryption step starting from the "Five block Feistel Cipher" to the first Vignère cipher to decrypt the encrypted text.
Figure 3.0 shows the console log of each encryption and decryption step for a sample message using this program, which demonstrates that the algorithm is fully reversible.

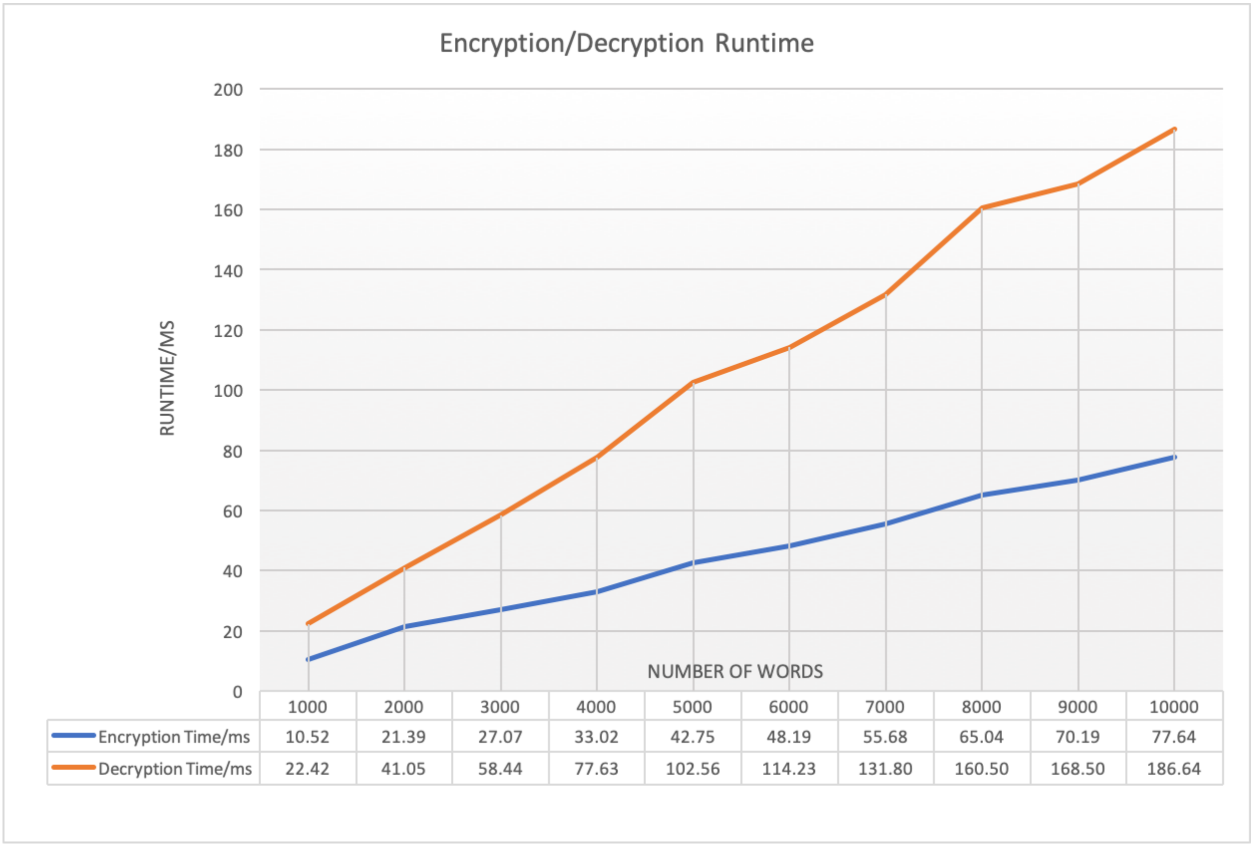


```
Message is:                    The tower of Babel is going to fall.
Key is:                        D3K7e%qk@y6Z%Ayb

Vigenere encryption(1):        O.G:wj=G-#j,a\d]+N:lnEI*li-a/rz,C'o)

S-BOX substitution:            4$b BYrb;sY]VkuFSP 7c/K,7A;V2v^]RQ@'

Vigenere encryption(2):        xWNW'-^: GE5;?a}8$knHb7c{t'.vJJ57%,^

P-BOX permutation:             N'WxW:G^- ;a5E?$n8}k7{bHc.J'tv7,5J%|||^|

Vigenere encryption(3):        s9cY|LS?R2GBZWKd4J*L\.n))@Vg:)ClZ\1]B/j]

Caesar Cipher:                 Jo:0S#*u)h}x1.";j!`#3dE__v->p_yC13g4xeA4

----5B custom Feistel encryption written to file-----

Please enter the .txt filename with the encrypted message:
encrypted.txt
---------------------------------------------
----------------DECRYPTION!------------------
---------------------------------------------


Please enter the key to decrypt the text:
D3K7e%qk@y6Z%Ayb
ENCRYPTED TEXT:                0100110000100101011101110111101101100010000000110110000000110111000010010010110010100011100110010011010010
100110000010011010101110010000000010011100001010000100100011001100101110000110000000001001101110010010000110011100110111010001110110111001000
11100001001011101000100100001010110000100110011001000000110000000101000000101

5B Feistel Decipher:           Jo:0S#*u)h}x1.";j!`#3dE__v->p_yC13g4xeA4

Caesar Decipher:               s9cY|LS?R2GBZWKd4J*L\.n))@Vg:)ClZ\1]B/j]

Vigenere Decipher(3):          N'WxW:G^- ;a5E?$n8}k7{bHc.J'tv7,5J%|||^|

Reverse P-BOX:                 xWNW'-^: GE5;?a}8$knHb7c{t'.vJJ57%,^

Vigenere Decipher(2):          4$b BYrb;sY]VkuFSP 7c/K,7A;V2v^]RQ@'

REVERSE S-BOX:                 O.G:wj=G-#j,a\d]+N:lnEI*li-a/rz,C'o)

Vigenere Decipher(1):          The tower of Babel is going to fall.

Decrypted message:             The tower of Babel is going to fall.
```

Figure 3.0: Console output of encryption algorithm on sample message.

Computational Requirements:

The runtime of encryption and decryption pair of the algorithm on texts of various lengths is shown in Figure 4.0. The algorithm was run on texts ranging from 1000 words to 10,000 words and the runtime of each was recorded and plotted in a graph.



	1000	2000	3000	4000	5000	6000	7000	8000	9000	10000
Encryption Time/ms	10.52	21.39	27.07	33.02	42.75	48.19	55.68	65.04	70.19	77.64
Decryption Time/ms	22.42	41.05	58.44	77.63	102.56	114.23	131.80	160.50	168.50	186.64

Figure 4.0: Encryption/Decryption runtime of algorithm on text of various lengths.

The text is randomly generated and it consists of all the different characters from the printable ASCII table. This includes alpha-numeric characters, punctuation marks, and spaces. As we can see in the figure, the runtime of both encryption and decryption is directly proportional to the length of the text. The decryption takes approximately twice as long as encryption and the difference gets larger as more text is taken into consideration. This highlights how a behavior similar to that of a one-way-trap is accomplished by the algorithm.

This implementation explores various cipher methods including Caesar's cipher, Vigènere cipher, the use of permutation and substitution boxes, and also a Feistel structure cipher. It is a fully functional symmetric encryption-decryption pair that uses a 128-bit key.

REFERENCES:

[1] William Stallings, W. (2005). Cryptography and Network Security. Principles and Practices (Fourth Edition). Prentice Hall.