RV COLLEGE OF ENGINEERING ®
BENGALURU-560059

(Autonomous Institution Affiliated to VTU, Belagavi)

"ENCRYPTION AND DECRYPTION USING SP Networks"

REPORT
For

Cryptography and Network Security Laboratory Project

(16IS73)
Submitted By

Sudarshan VM (1RV17IS049), Utkarsh Singh (1RV17IS054)

Under the Guidance of
Prof. Geetha V.
Assistant Prof, Dept. of ISE

in partial fulfillment for the award of degree
of
Bachelor of Engineering
in

INFORMATION SCIENCE AND ENGINEERING
Sept-Dec 2020

RV COLLEGE OF ENGINEERING®, BENGALURU – 560059

(Autonomous Institution Affiliated to VTU, Belagavi)

DEPARTMENT OF INFORMATION SCIENCE AND ENGINEERING

CERTIFICATE

Certified that the Mini Project work entitled '**Encryption and decryption using SP Networks**' has been carried out as a part of Cryptography and Network Security (**16IS73)** in partial fulfilment for the award of the degree of Bachelor of Engineering in Information Science and Engineering of the Visvesvaraya Technological University, Belagavi during the year 2020-2021 by **UTKARSH SINGH (1RV17IS054), SUDARSHAN VM (1RV17IS049),**who are bonafide students of **R V College of Engineering®** , Bengaluru. It is certified that all the corrections/suggestions indicated for the internal assessment have been incorporated in the report deposited in the departmental library. The report has been approved as it satisfies the academic requirements in respect of work prescribed by the institution for the said degree.

Prof. Geetha V	**Dr. Sagar B. M.**
Assistant Professor	**Head Of Department**
Department of ISE,	Department of ISE,
RVCE, Bengaluru-59	RVCE, Bengaluru-59

Name of Examiners	Signature
1.________________	________________
2.________________	________________

TABLE OF CONTENTS

1. INTRODUCTION

1.1 Purpose

The purpose of this project is to develop a system to protect data transmitted in the likely presence of an adversary. A cryptography transformation of data is a procedure by which plain text data is disguised, or encrypted, resulting in an altered text, called **ciphertext** or **encrypted text**, that does not reveal the original input. The ciphertext can be reverse-transformed by a designated recipient, called **Decrypted** or **deciphered text**. The purpose is to enable one to store sensitive information or transmit it across insecure networks (like the Internet) so that it cannot be read by anyone except the intended recipient.

1.2 Motivation

There are several reasons why we choose to select this project.

- First and foremost, since we have this subject as part of a curriculum, taking up a project will help us gain a better understanding about the subject theoretically as well as practically.
- As we are taking up a project in our final semester, this mini project will help us gain a practical perspective of how a project is developed and what and all are the tasks that need to be carried out in successfully completing the project along with the documentation.
- To build a safe and secure system that enables transmission of sensitive data between sender and receiver.

1.3 OVERVIEW

Encryption techniques come in many shapes and forms. There are countless algorithms starting from Caesar ciphers to AES. Most of the advanced algorithms use a variety of techniques in combination to achieve highly complex and hard-to-break algorithms. One such algorithm is the **SP Network Algorithm**.

In cryptography, an **SP-network,** or **substitution–permutation network (SPN)**, is a series of linked mathematical operations used in block cipher algorithms such as AES (Rijndael), 3-Way, Kalyna, Kuznyechik, PRESENT, SAFER, SHARK, and Square.

Such a network takes a block of the plaintext and the key as inputs, and applies several alternating "rounds" or "layers" of substitution boxes (S-boxes) and permutation boxes (P-boxes) to produce the ciphertext block. The S-boxes and P-boxes transform (sub-)blocks of input bits into output bits. It is common for these transformations to be operations that are efficient to perform in hardware, such as exclusive or (XOR) and bitwise rotation. The key is introduced in each round, usually in the form of "round keys" derived from it. (In some designs, the S-boxes themselves depend on the key.)

1.4 PROBLEM STATEMENT

This project is an implementation of Substitution–permutation network(SP) cryptosystem which includes encryption, decryption. So, it can cipher the message texted then can decipher like a receiver.

2. SOFTWARE REQUIREMENT SPECIFICATION

Problem Statement:

“To build an encryption and decryption algorithm using substitution and permutation cipher (SP) algorithms in python.”

Functional Requirements :

- The software should be able to produce encryption for the given input plain text.
- The encryption key should be unique for every stage of execution.
- The encryption process should be sufficiently fast and lightweight.
- The software should be able to produce decryption for the given input plain text.

Non-Functional requirements:

- Performance: Relatively fast encryption/decryption depending on string length.
- Portability: The software must support multiple operating systems.

Hardware requirements:

- Processor: Pentium 4.0 or higher/ Amd Athlon x2 or higher
- RAM: 2 gb or more
- Hard Drive: 1 GB or more

Software Requirements:

- Operating System:
    1. Window Vista, Windows 7 or above.
    2. Ubuntu 16.04 LTS or above
    3. A python interpreter
- Development tools used:

3. Language: python 3.7+
4. Libraries: pandas, numpy, tkinter
5. Development platform: Jupyter notebook.

3. METHODOLOGY

Key transformations

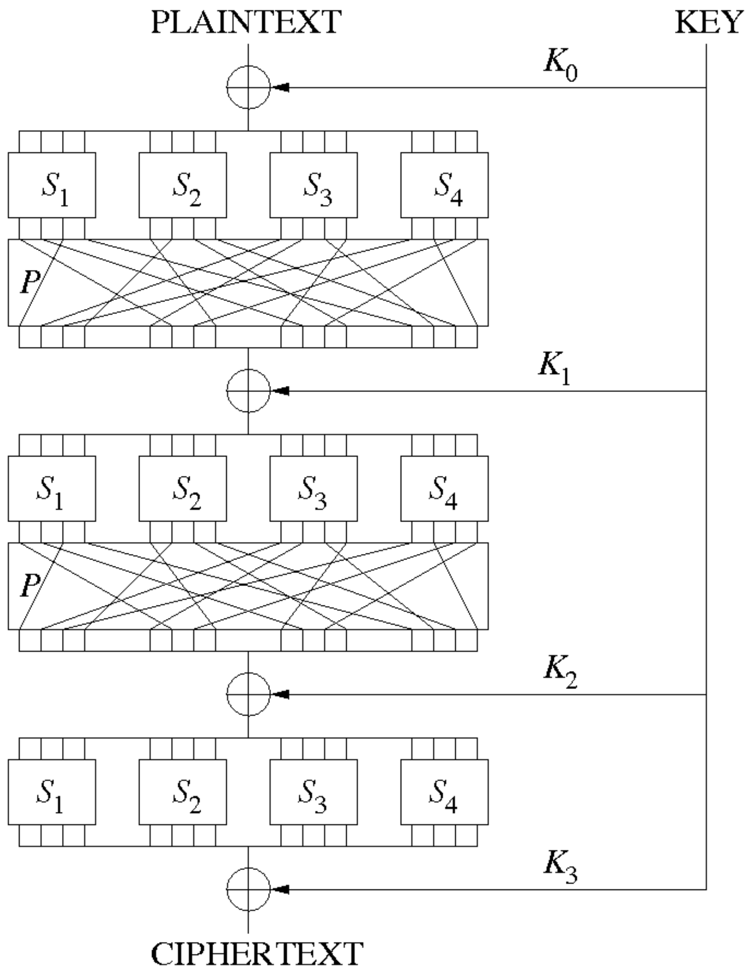
There are 2 keys used in this algorithm. First key, k0 is not used directly in encryption. The key k1 is used to encrypt the data once. After that key k0 is used to encrypt the key k1 which gives us k2. This is done using substitution cipher. So the key stays the same length as before and there is no redundant data. Same way we get key k3 by encrypting the key k2 with key k1. This technique is called cipher block-chaining where you use the encrypted data to re encrypt the next block of data. This is usually performed with XOR operations but it can be used in this manner also. In this way we have a list of kn keys which can be used to encrypt the data any number of times we desire. The initial key for the entire algorithm is the string addition of k0 and k1.

plain_text -> Permutation cipher -> Substitution cipher -> encrypted_text
is repeated n times, where 'n' is user input.

In the Permutation cipher, we use the keys as prime numbers per stage. For example, 1st stage uses 2, second stage uses 3, then third stage uses 5 likewise.

In the Substitution cipher, we use a different key that we computed before, those are k1, k2, k3 etc.

4. ALGORITHM AND DESIGN



A Diagrammatic representation of the SP- Network of 3 rounds

<u>Encryption</u>

Our algorithm works in 2 sub-stages. In the first sub-stage we run the initial input text through the Permutation cipher. Then we use this output as an input to the Substitution cipher. These 2 sub-stages make up one stage of our algorithm.

The algorithm

plain_text -> Permutation cipher -> Substitution cipher -> encrypted_text
is repeated n times, where 'n' is user input.

In the Permutation cipher, we use the keys as prime numbers per stage. For example, 1st stage uses 2, second stage uses 3, then third stage uses 5 likewise.

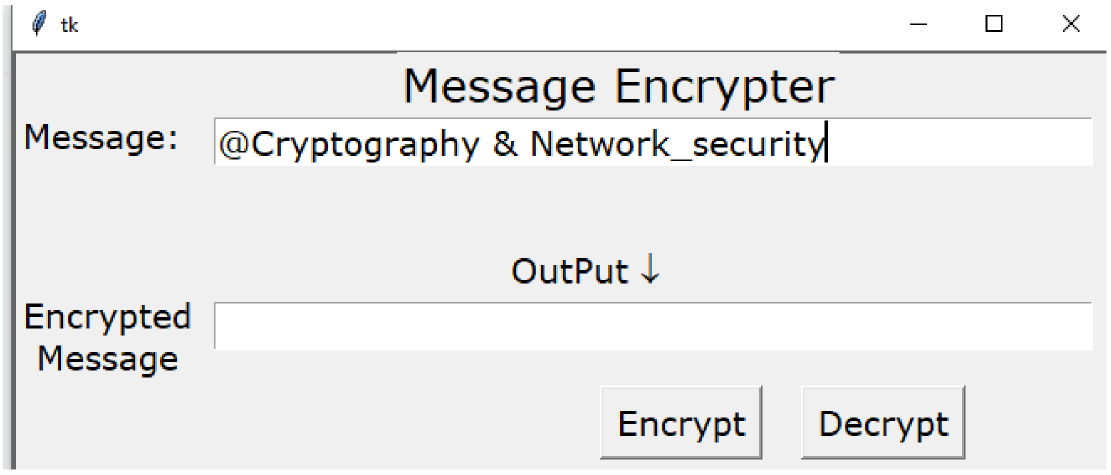
In the Substitution cipher, we use a different key that we computed before, those are k1, k2, k3 etc.

Decryption

Decryption is done by reversing this process. The information we need for decryption is the key, number of stages, and the length of initial plain text input for calculating the padding that was added in the Permutation cipher.

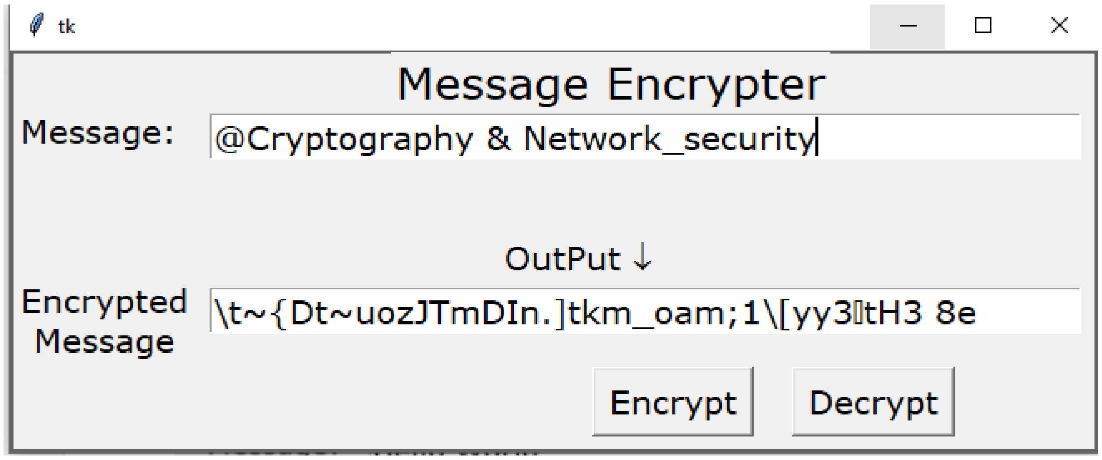
5. RESULT

1. GUI



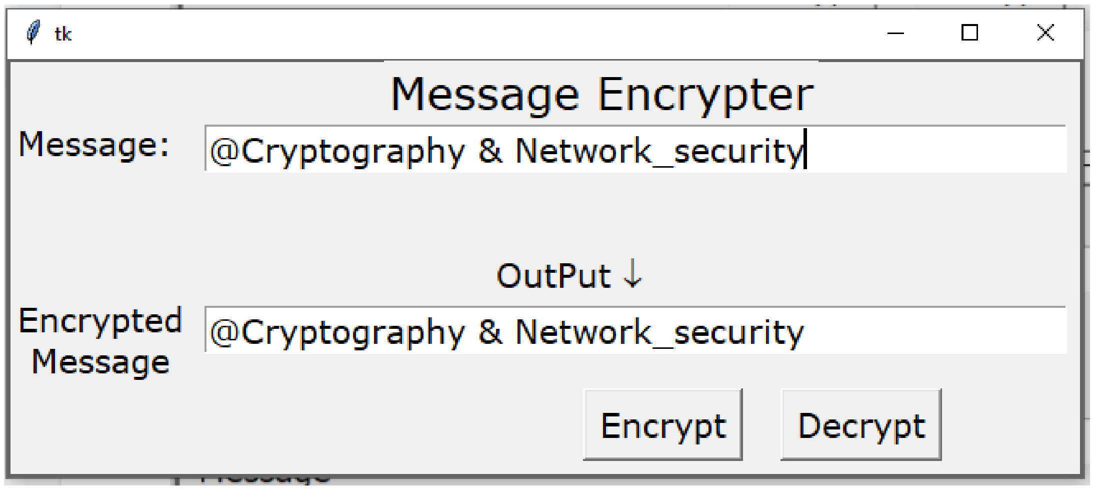
Message: @Cryptography & Network_security

Encrypted Message:



Message: @Cryptography & Network_security

Encrypted Message: \t~{Dt~uozJTmDIn.]tkm_oam;1\[yy3▯tH3 8e

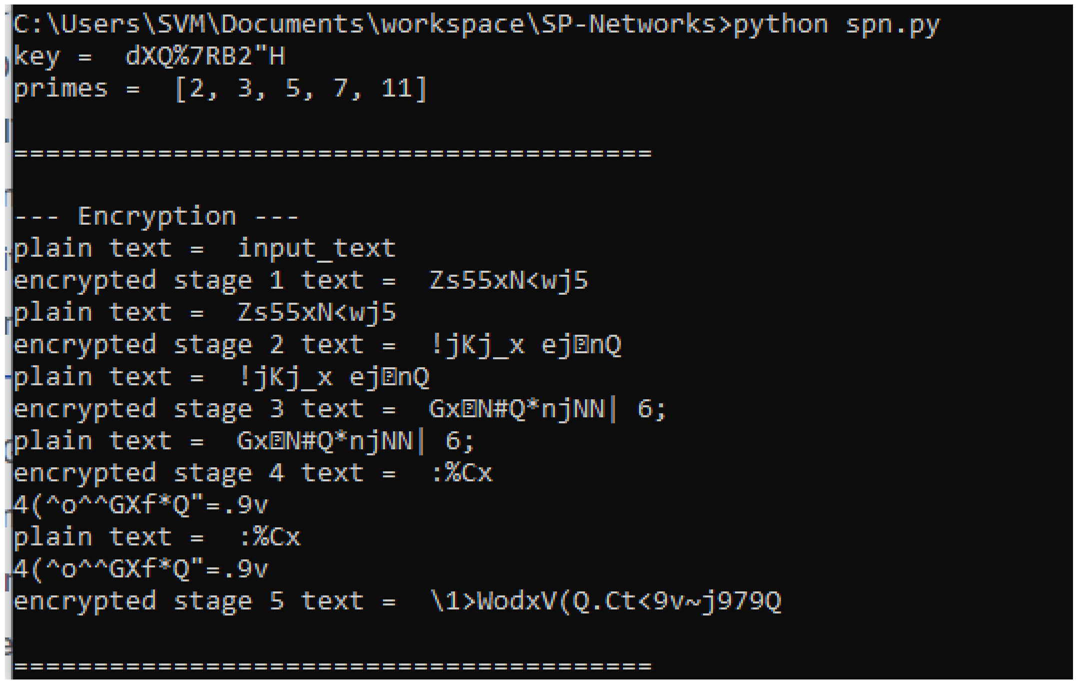


Message: @Cryptography & Network_security

Encrypted Message: @Cryptography & Network_security

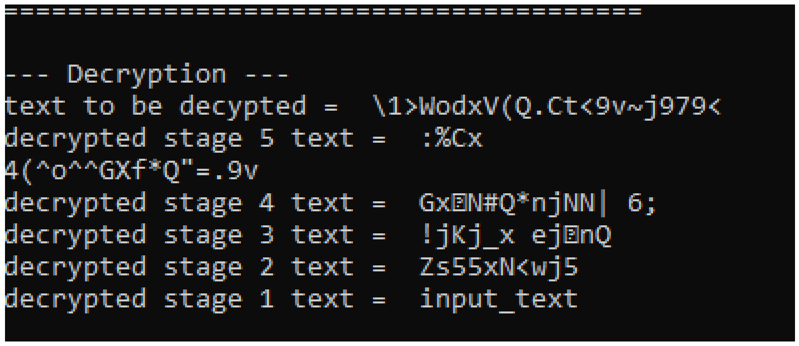
2. COMMAND LINE

Encrypted message at each stage:



```
C:\Users\SVM\Documents\workspace\SP-Networks>python spn.py
key =  dXQ%7RB2"H
primes =  [2, 3, 5, 7, 11]

========================================

--- Encryption ---
plain text =  input_text
encrypted stage 1 text =  Zs55xN<wj5
plain text =  Zs55xN<wj5
encrypted stage 2 text =  !jKj_x ej▯nQ
plain text =  !jKj_x ej▯nQ
encrypted stage 3 text =  Gx▯N#Q*njNN| 6;
plain text =  Gx▯N#Q*njNN| 6;
encrypted stage 4 text =  :%Cx
4(^o^^GXf*Q"=.9v
plain text =  :%Cx
4(^o^^GXf*Q"=.9v
encrypted stage 5 text =  \1>WodxV(Q.Ct<9v~j979Q

========================================
```

Decrypted message at each stage:



```
========================================

--- Decryption ---
text to be decypted =  \1>WodxV(Q.Ct<9v~j979<
decrypted stage 5 text =  :%Cx
4(^o^^GXf*Q"=.9v
decrypted stage 4 text =  Gx▯N#Q*njNN| 6;
decrypted stage 3 text =  !jKj_x ej▯nQ
decrypted stage 2 text =  Zs55xN<wj5
decrypted stage 1 text =  input_text
```

6.TESTING

Testing Methodologies
A strategy for software testing must accommodate low-level tests that are necessary to verify that a small source code segment has been correctly implemented as well as high-level tests that validate major system functions against customer requirements. A strategy must provide guidance for the practitioner and a set of milestones for the manager. Because the steps of the test strategy occur at a time when deadline pressure begins to rise, progress must be measurable and problems must surface as early as possible. Following testing techniques are well known and the same strategy is adopted during this project testing.

Unit Testing
Unit testing refers to tests that verify the functionality of a specific section of code, usually at the function level or at the class level. Small independent parts of code are tested for its right execution. These types of tests are usually written by developers as they work on code and can be done manually, to ensure that the specific function is working as expected.
Since this is a cryptography algorithm which may have various forms of user input, it would be nearly impossible to test every possible user combination. However, testing functionality and likely user scenarios should be our overall goal of testing this software. Unit testing focuses on verification effort on the smallest unit of software design- the software component or module. The unit test is white-box oriented.

Integration Testing
First and most importantly we will ensure the program compiles and runs without errors. Both encryption and decryption must function properly after integration.

System Testing
System testing is actually a series of different tests whose primary purpose is to fully exercise the computer-based system. Two types of testing have been described which have been taken for this project. It is to check all modules worked on input basis. System testing can be stated as the process of validating and verifying that a computer program / application / product meets the requirements specified in the system requirements specifications (SRS) document.

Test Case ID	Description	Expected Output	Actual Output	Result
TC01	Compilation and execution when user runs the	Successful compilation and execution.	Successful compilation and execution.	Pass

	program			
TC02	Input only digits as plain text (before configuring it for digits+letters+ punctuations)	Encrypted message after last stage of sp network	Error (only letters input is supported)	Fail (passed after allowing digits+letters+punctuations)
TC03	Input only symbols as plain text (before configuring it for digits+letters+ punctuations)	Encrypted message after last stage of sp network	Error (only letters input is supported)	Fail (passed after allowing digits+letters+punctuations)
TC04	Input symbols and digits both as plain text.	Encrypted message after last stage of sp network	Same as expected	pass
TC05	Input only digits as plain text (after configuring it for digits+letters+ punctuations)	Encrypted message after last stage of sp network	Same as expected	pass
TC06	Input only symbols as plain text (before configuring it for digits+letters+ punctuations)	Encrypted message after last stage of sp network	Same as expected	pass
TC07	Input only alphabet letters	Encrypted message after	same as expected	pass

	as plain text	last stage of sp network		
TC08	Input letters+ digits combination as plain text	Encrypted message after last stage of sp network	Same as expected	pass
TC09	Input letters+symbols as plain text	Encrypted message after last stage of sp network	Same as expected	pass
TC10	Input encrypted message as plain text	Successful decryption and proper display of plain text	Same as expected	pass
TC11	Input message other than encrypted text	Successful decryption but decrypted message not same as plain text	Same as expected	pass

7. CONCLUSION

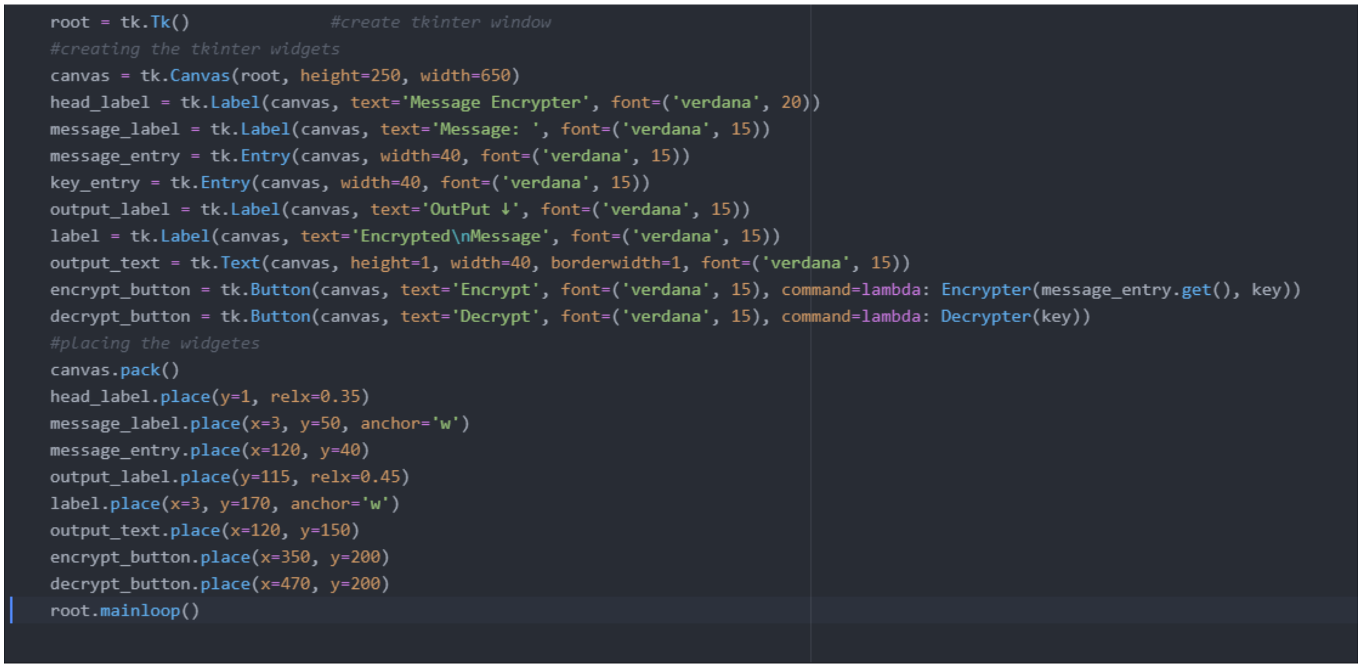
Substitution cipher and Transposition cipher when used together are referred to as SPN (Substitution-Permutation Network). It uses complex algorithms to do encryption and decryption which makes it more secure.

8. REFERENCES

1. https://en.wikipedia.org/wiki/Substitution-permutation_network
2. http://barrywatson.se/crypto/crypto_sp_network.html

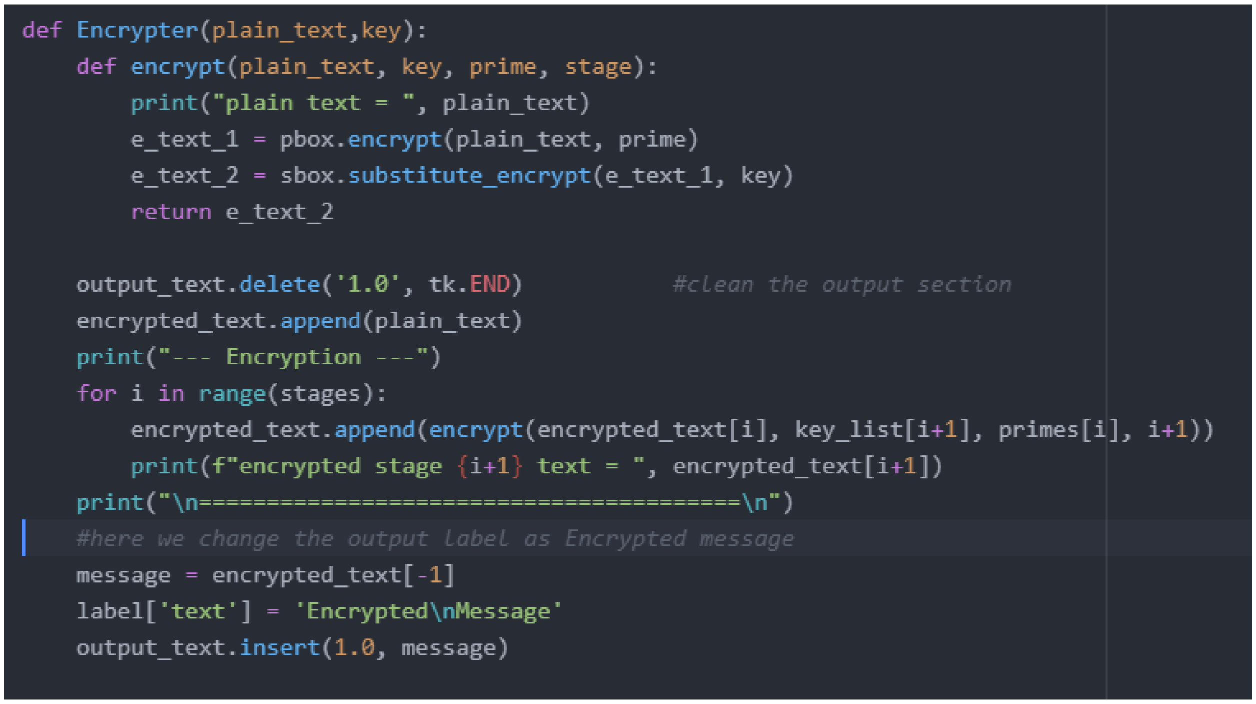
9. APPENDIX (CODE SNIPPETS)

GUI



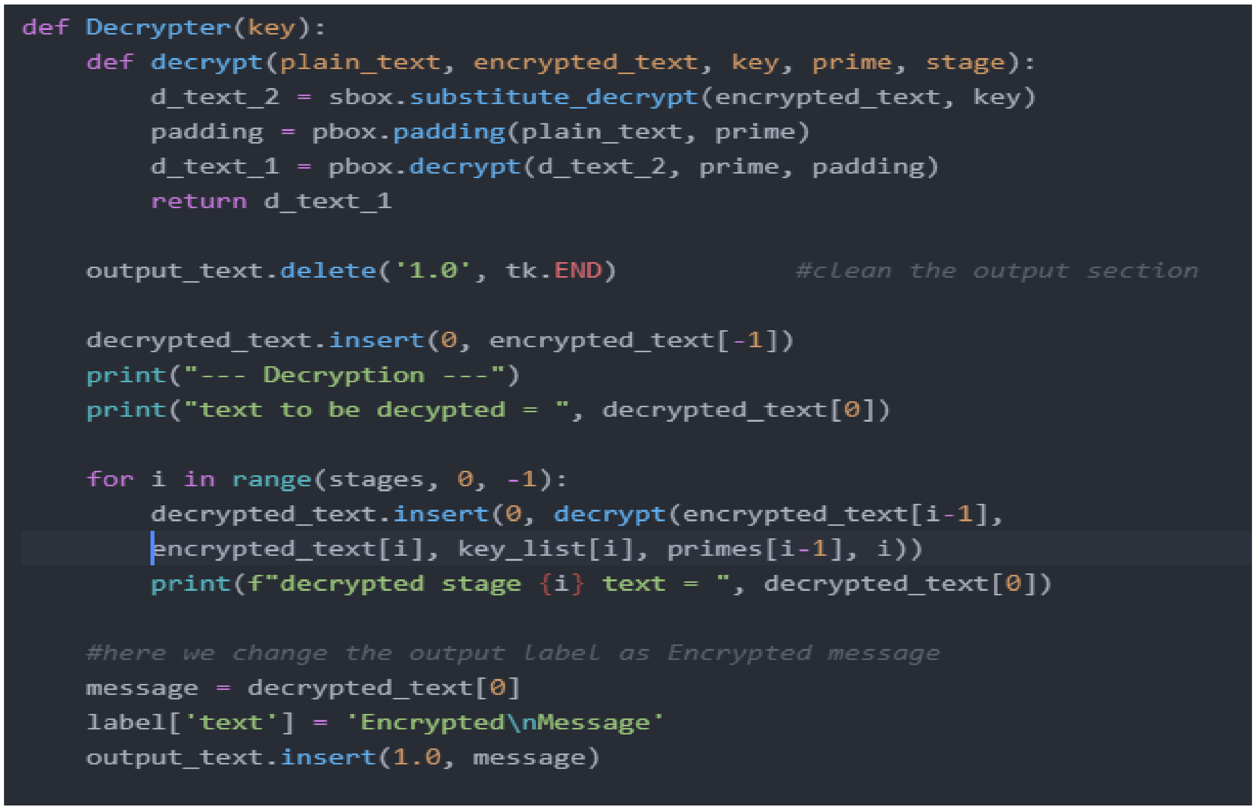
```
root = tk.Tk()              #create tkinter window
#creating the tkinter widgets
canvas = tk.Canvas(root, height=250, width=650)
head_label = tk.Label(canvas, text='Message Encrypter', font=('verdana', 20))
message_label = tk.Label(canvas, text='Message: ', font=('verdana', 15))
message_entry = tk.Entry(canvas, width=40, font=('verdana', 15))
key_entry = tk.Entry(canvas, width=40, font=('verdana', 15))
output_label = tk.Label(canvas, text='OutPut ↓', font=('verdana', 15))
label = tk.Label(canvas, text='Encrypted\nMessage', font=('verdana', 15))
output_text = tk.Text(canvas, height=1, width=40, borderwidth=1, font=('verdana', 15))
encrypt_button = tk.Button(canvas, text='Encrypt', font=('verdana', 15), command=lambda: Encrypter(message_entry.get(), key))
decrypt_button = tk.Button(canvas, text='Decrypt', font=('verdana', 15), command=lambda: Decrypter(key))
#placing the widgetes
canvas.pack()
head_label.place(y=1, relx=0.35)
message_label.place(x=3, y=50, anchor='w')
message_entry.place(x=120, y=40)
output_label.place(y=115, relx=0.45)
label.place(x=3, y=170, anchor='w')
output_text.place(x=120, y=150)
encrypt_button.place(x=350, y=200)
decrypt_button.place(x=470, y=200)
root.mainloop()
```

ENCRYPTER



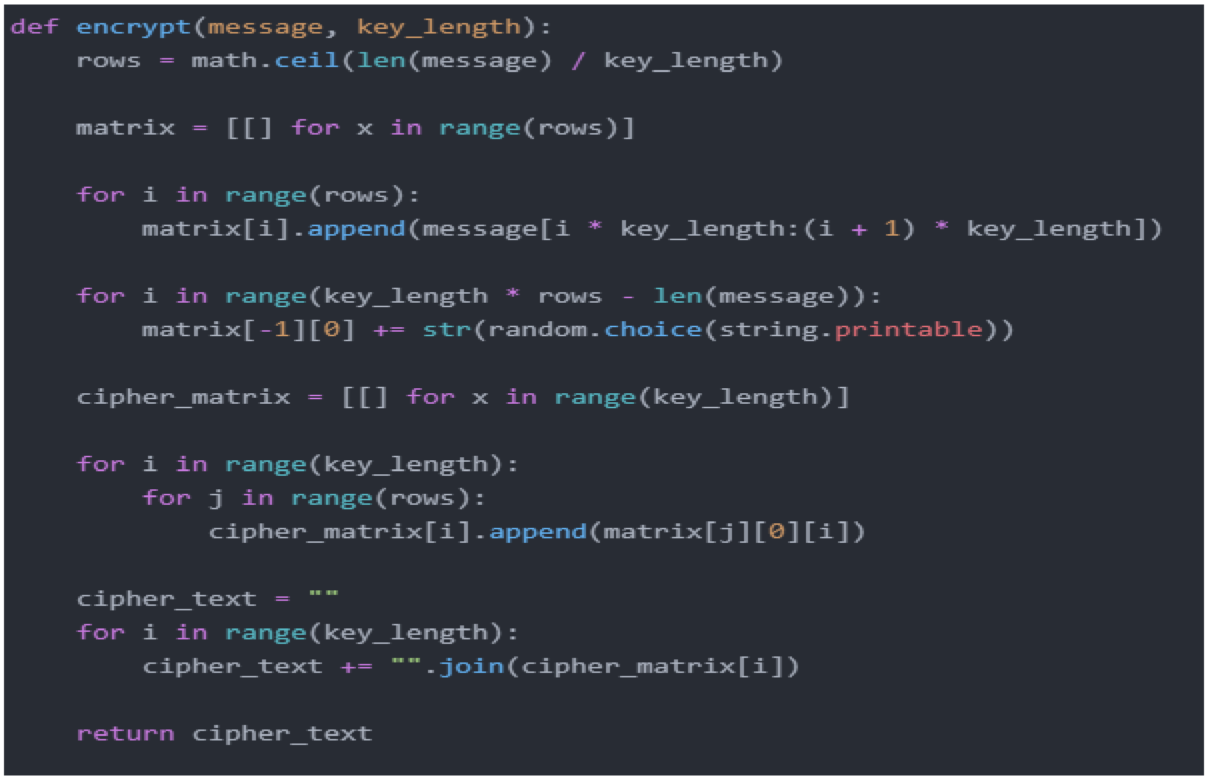
```
def Encrypter(plain_text,key):
    def encrypt(plain_text, key, prime, stage):
        print("plain text = ", plain_text)
        e_text_1 = pbox.encrypt(plain_text, prime)
        e_text_2 = sbox.substitute_encrypt(e_text_1, key)
        return e_text_2

    output_text.delete('1.0', tk.END)          #clean the output section
    encrypted_text.append(plain_text)
    print("--- Encryption ---")
    for i in range(stages):
        encrypted_text.append(encrypt(encrypted_text[i], key_list[i+1], primes[i], i+1))
        print(f"encrypted stage {i+1} text = ", encrypted_text[i+1])
    print("\n========================================\n")
    #here we change the output label as Encrypted message
    message = encrypted_text[-1]
    label['text'] = 'Encrypted\nMessage'
    output_text.insert(1.0, message)
```

DECRYPTER:



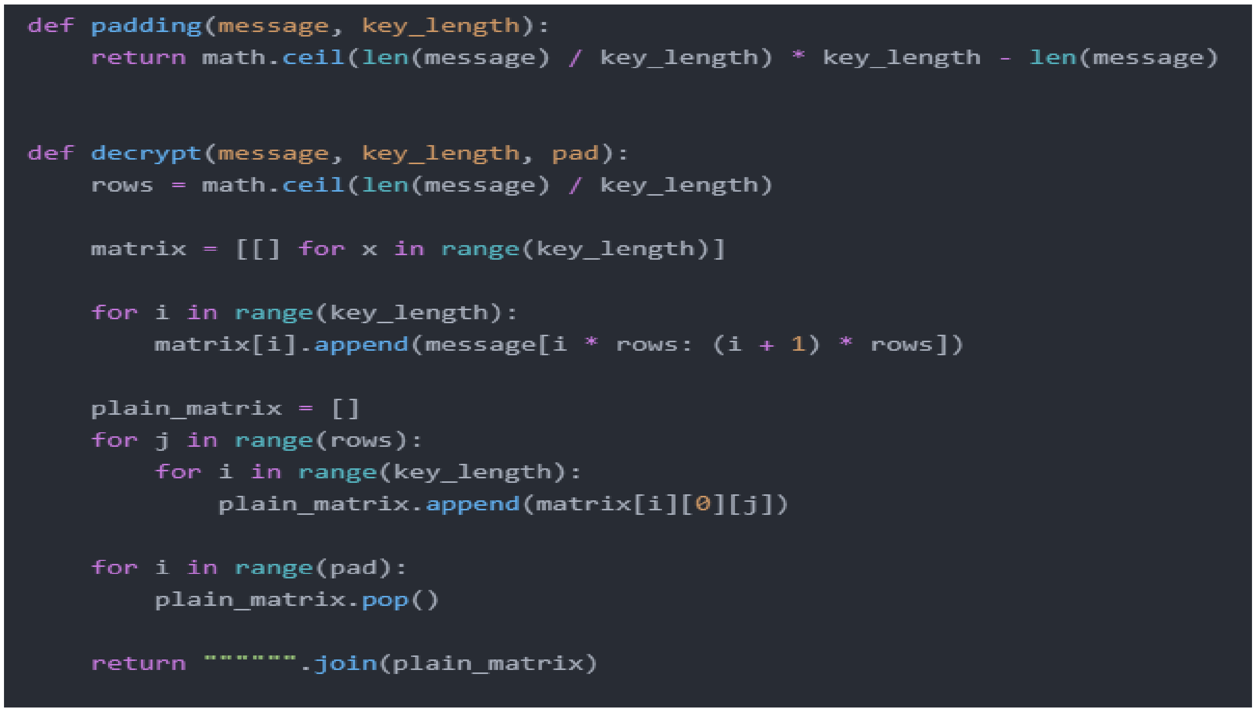
```
def Decrypter(key):
    def decrypt(plain_text, encrypted_text, key, prime, stage):
        d_text_2 = sbox.substitute_decrypt(encrypted_text, key)
        padding = pbox.padding(plain_text, prime)
        d_text_1 = pbox.decrypt(d_text_2, prime, padding)
        return d_text_1

    output_text.delete('1.0', tk.END)           #clean the output section

    decrypted_text.insert(0, encrypted_text[-1])
    print("--- Decryption ---")
    print("text to be decypted = ", decrypted_text[0])

    for i in range(stages, 0, -1):
        decrypted_text.insert(0, decrypt(encrypted_text[i-1],
        encrypted_text[i], key_list[i], primes[i-1], i))
        print(f"decrypted stage {i} text = ", decrypted_text[0])

    #here we change the output label as Encrypted message
    message = decrypted_text[0]
    label['text'] = 'Encrypted\nMessage'
    output_text.insert(1.0, message)
```

P-BOX ENCRYPTION:



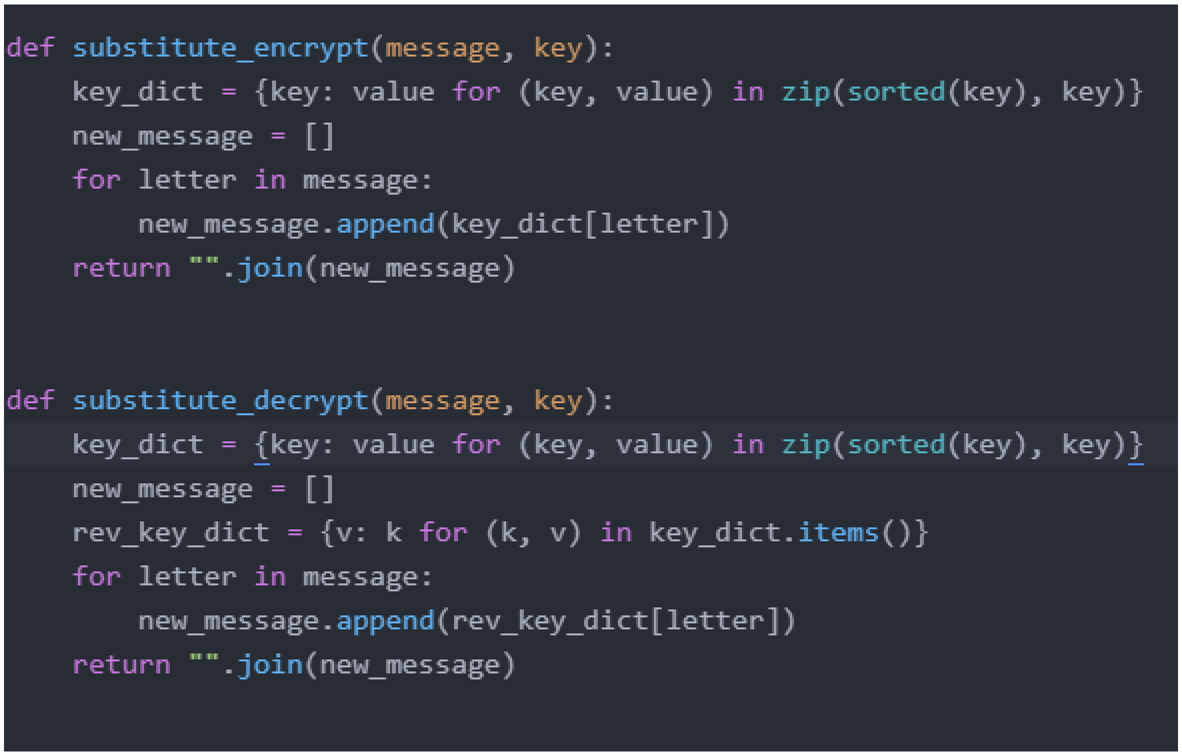
```
def encrypt(message, key_length):
    rows = math.ceil(len(message) / key_length)

    matrix = [[] for x in range(rows)]

    for i in range(rows):
        matrix[i].append(message[i * key_length:(i + 1) * key_length])

    for i in range(key_length * rows - len(message)):
        matrix[-1][0] += str(random.choice(string.printable))

    cipher_matrix = [[] for x in range(key_length)]

    for i in range(key_length):
        for j in range(rows):
            cipher_matrix[i].append(matrix[j][0][i])

    cipher_text = ""
    for i in range(key_length):
        cipher_text += "".join(cipher_matrix[i])

    return cipher_text
```

P-BOX DECRYPTION:



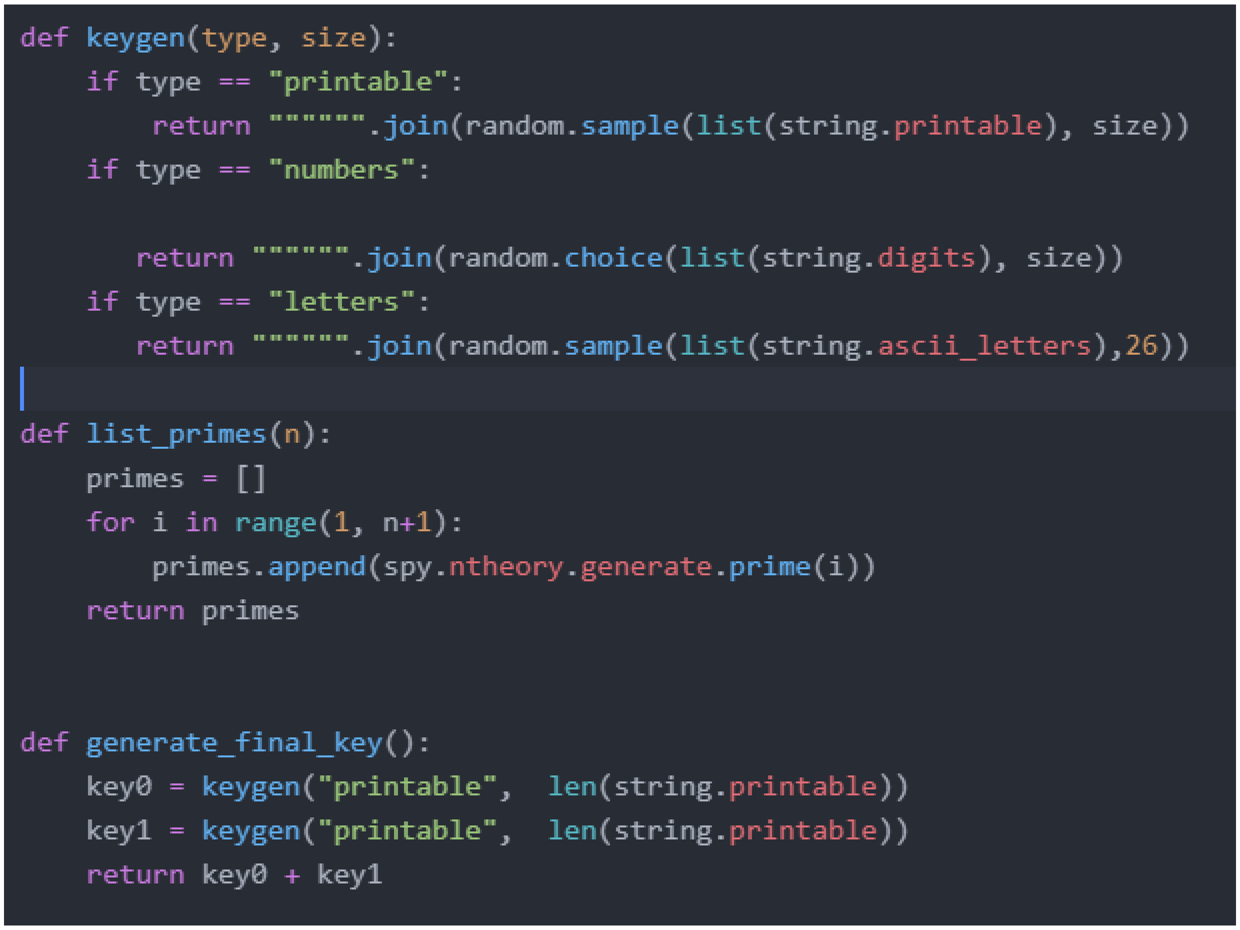
```
def padding(message, key_length):
    return math.ceil(len(message) / key_length) * key_length - len(message)


def decrypt(message, key_length, pad):
    rows = math.ceil(len(message) / key_length)

    matrix = [[] for x in range(key_length)]

    for i in range(key_length):
        matrix[i].append(message[i * rows: (i + 1) * rows])

    plain_matrix = []
    for j in range(rows):
        for i in range(key_length):
            plain_matrix.append(matrix[i][0][j])

    for i in range(pad):
        plain_matrix.pop()

    return """""".join(plain_matrix)
```

S-BOX ENCRYPTION & DECRYPTION:



```
def substitute_encrypt(message, key):
    key_dict = {key: value for (key, value) in zip(sorted(key), key)}
    new_message = []
    for letter in message:
        new_message.append(key_dict[letter])
    return "".join(new_message)


def substitute_decrypt(message, key):
    key_dict = {key: value for (key, value) in zip(sorted(key), key)}
    new_message = []
    rev_key_dict = {v: k for (k, v) in key_dict.items()}
    for letter in message:
        new_message.append(rev_key_dict[letter])
    return "".join(new_message)
```

KEY-GENERATION:



```
def keygen(type, size):
    if type == "printable":
        return """""".join(random.sample(list(string.printable), size))
    if type == "numbers":

       return """""".join(random.choice(list(string.digits), size))
    if type == "letters":
       return """""".join(random.sample(list(string.ascii_letters),26))

def list_primes(n):
    primes = []
    for i in range(1, n+1):
        primes.append(spy.ntheory.generate.prime(i))
    return primes


def generate_final_key():
    key0 = keygen("printable",  len(string.printable))
    key1 = keygen("printable",  len(string.printable))
    return key0 + key1
```