**Experiment No. 3**

**Aim:** To implement Additive Cipher (Caesar Cipher).

**Theory: Additive Cipher** also known as **Caesar Cipher** is an example of **Substitution Cipher** in which each character of Plain text is replaced by some other character present in its domain using key in the range of size of domain, in Additive Cipher size of domain is equal to total alphabets i.e. 26 (count starts from 0 but we exclude 0 because it gives same cipher text as plain text).

**Additive Cipher** key range**:**

**Encipher Function:** – {Where, is the numeric value of alphabet in Plain text and k is the key agreed by sender and receiver}.

**Decipher Function:** – {Where, is the numeric value of alphabet in Cipher text and k is the same key as in Encipher Function}.

We mod by 26 because the numeric values of our domain can never exceed 25. Cipher text is always in capital case.

Alphabets and their numeric values: Values are same for both Cipher text and Plain text.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z |
| a | b | c | d | e | f | g | h | i | j | k | l | m | n | o | p | q | r | s | t | u | v | w | x | y | z |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 |

E.g. Plain text: Python is awesome, Key = 5

Similarly, doing for each this for each alphabet in Plain text we get Cipher text as: UDYMTSNXFBJXTRJ. We removed spaces in Plain text we can also keep it as it is.

Cipher text: UDYMTSNXFBJXTRJ, Key = 5

Similarly, doing this for each alphabet in Cipher text we get Plain text as: pythonisawesome. It can be seen that it is same as above Plain text (spaces removed).

By above example it is clear that and. Since, both function are invertible they form one-to-one correspondence with each other.

**Implementation:**

import numpy as np

import string

class AdditiveCipher:

def \_\_init\_\_(self, key = np.random.randint(1,26)):

assert 1<=key<= 25, "Key should be in between [1,25]"

self.domain = 26

self.key = key

self.map = {key:value for key,value in zip(list(string.ascii\_uppercase),range(0,26))}

self.reverse\_map = {value:key for key,value in self.map.items()}

def encode(self, plain\_txt):

plain\_txt = plain\_txt.replace(" ","")

assert plain\_txt.isalpha() or plain\_txt, "Plain text should only contain alphabetic character"

cipher\_txt = ""

for txt in plain\_txt:

cipher\_txt += self.reverse\_map[(self.map[txt.upper()]+self.key)%self.domain]

return cipher\_txt

def decode(self, cipher\_txt):

assert cipher\_txt.isupper() and cipher\_txt.isalpha(), "Cipher text should only contain uppercase alphabetic character"

plain\_txt = ""

for txt in cipher\_txt:

plain\_txt += self.reverse\_map[(self.map[txt]-self.key)%self.domain].lower()

return plain\_txt

additive = AdditiveCipher(int(input("Enter key in between [1,26] for encipherment and decipherment: ")))

print()

z = additive.encode(input("Enter a text [a-z]|[A-Z] to encode it:"))

y = additive.decode(z)

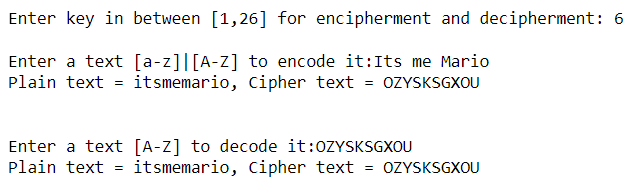
print(f"Plain text = {y}, Cipher text = {z}\n\n")

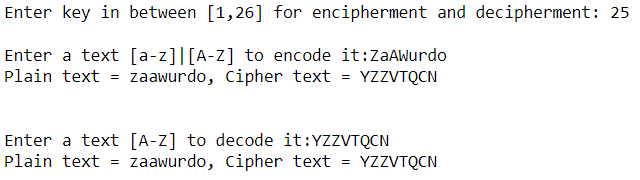
z = additive.decode(input("Enter a text [A-Z] to decode it:"))

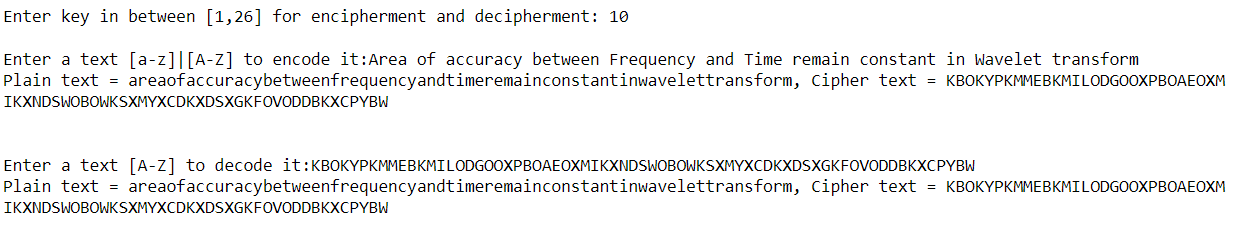
y = additive.encode(z)

print(f"Plain text = {z}, Cipher text = {y}")

**Output:**

****

****

****