MALAD KANDIVALI EDUCATION SOCIETY'S
NAGINDAS KHANDWALA COLLEGE OF COMMERCE,
ARTS & MANAGEMENT STUDIES & SHANTABEN NAGINDAS
KHANDWALA COLLEGE OF SCIENCE
MALAD [W], MUMBAI – 64
(AUTONOMOUS)

(Reaccredited 'A' Grade by NAAC)
(AFFILIATED TO UNIVERSITY OF MUMBAI)
(ISO 9001:2015)

CERTIFICATE

Name: Mr. kuldeep sushil patel

Roll No: 574 **Programme**: BSc CS **Semester**: V

This is certified to be a bonafide record of practical works done by the above student in the college laboratory for the course **Information and Network Security** (Course Code: **1854UCSPR**) for the partial fulfillment of Fifth Semester of BSc CS during the academic year 2020-2021.

The journal work is the original study work that has been duly approved in the year 2020-2021 by the undersigned.

External Examiner

Prof. Elizabeth Leah George
(Subject-In-Charge)

Date of Examination:

(College Stamp)

Name: Mr. kuldeep sushil patel **Roll No: 574**

Subject: Information and Network Security (1854UCSPR)

Class: T.Y.B.SC (CS) SEMESTER-V

NO	DATE	TITLE	SIGN
1.	16/07/2020	**IMPLEMENTING SUBSTITUTION CIPHER** 1. CAESAR CIPHER 2. MODIFIED CAESAR CIPHER 3. MONO-ALPHABETIC 4. POLY-ALPHABETIC (VERNAM)	
2.	30/07/2020	**IMPLEMENTING TRANSPOSITION CIPHER** 1. RAILFENCE TRANSPOSITION CIPHER 2. SIMPLE COLUMNAR TRANSPOSITION	
3.	13/08/2020	IMPLEMENTING DIFFIE-HELLMAN KEY EXCHANGE ALGORITHM	
4.	27/08/2020	1. IMPLEMENTING DES ALGORITHM 2. IMPLEMENTING AES ALGORITHM	
5.	10/09/2020	IMPLEMENTING RSA ALGORITHM	
6.	17/09/2020	MESSAGE DIGEST (MD5)	
7.	24/09/2020	HASH FUNCTION (HMAC SHA)	
8.	01/10/2020	1. ASYMMETRIC KEY ALGORITHM FOR PUBLIC PRIVATE KEY CRYPTOGRAPHY 2. SYMMETRIC KEY ALGORITHMS	
9.	15/10/2020	GENERATING PASS PHRASE FOR ENCRYPTION	

Practical 1

Aim : implementing the substitution cipher.

1] CAESOR CIPHER

Theory

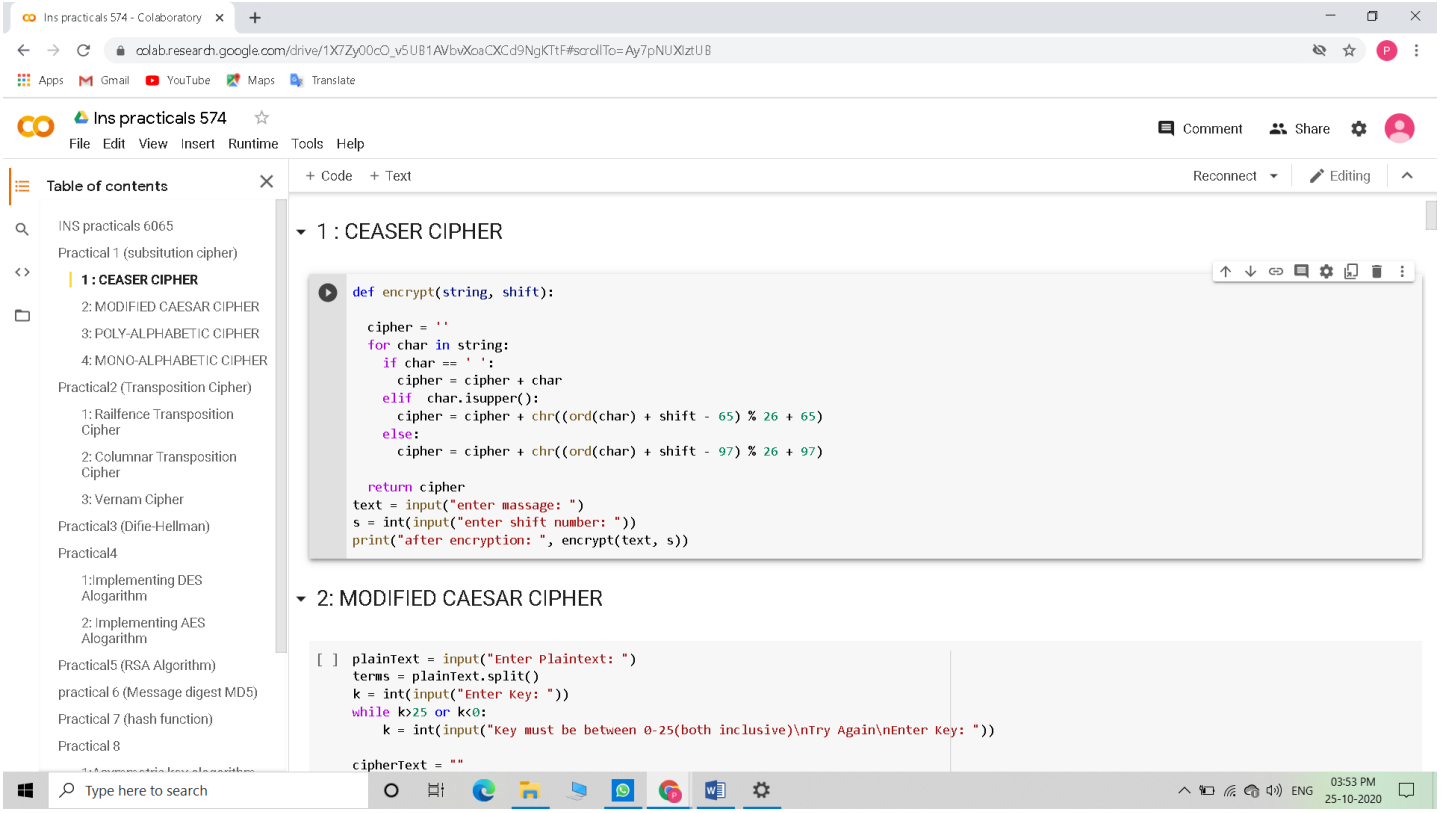
- The Caesar Cipher technique is one of the earliest and simplest method of encryption technique.
- It's simply a type of substitution cipher, i.e., each letter of a given text is replaced by a letter some fixed number of positions down the alphabet.
- For example with a shift of 1, A would be replaced by B, B would become C, and so on.
- The method is apparently named after Julius Caesar, who apparently used it to communicate with his officials.
- Thus to cipher a given text we need an integer value, known as shift which indicates the number of position each letter of the text has been moved down.
- the formula of Caesar cipher is
  for encryption phase with shift n

$$E_n(x) = (x+n) \bmod 26$$

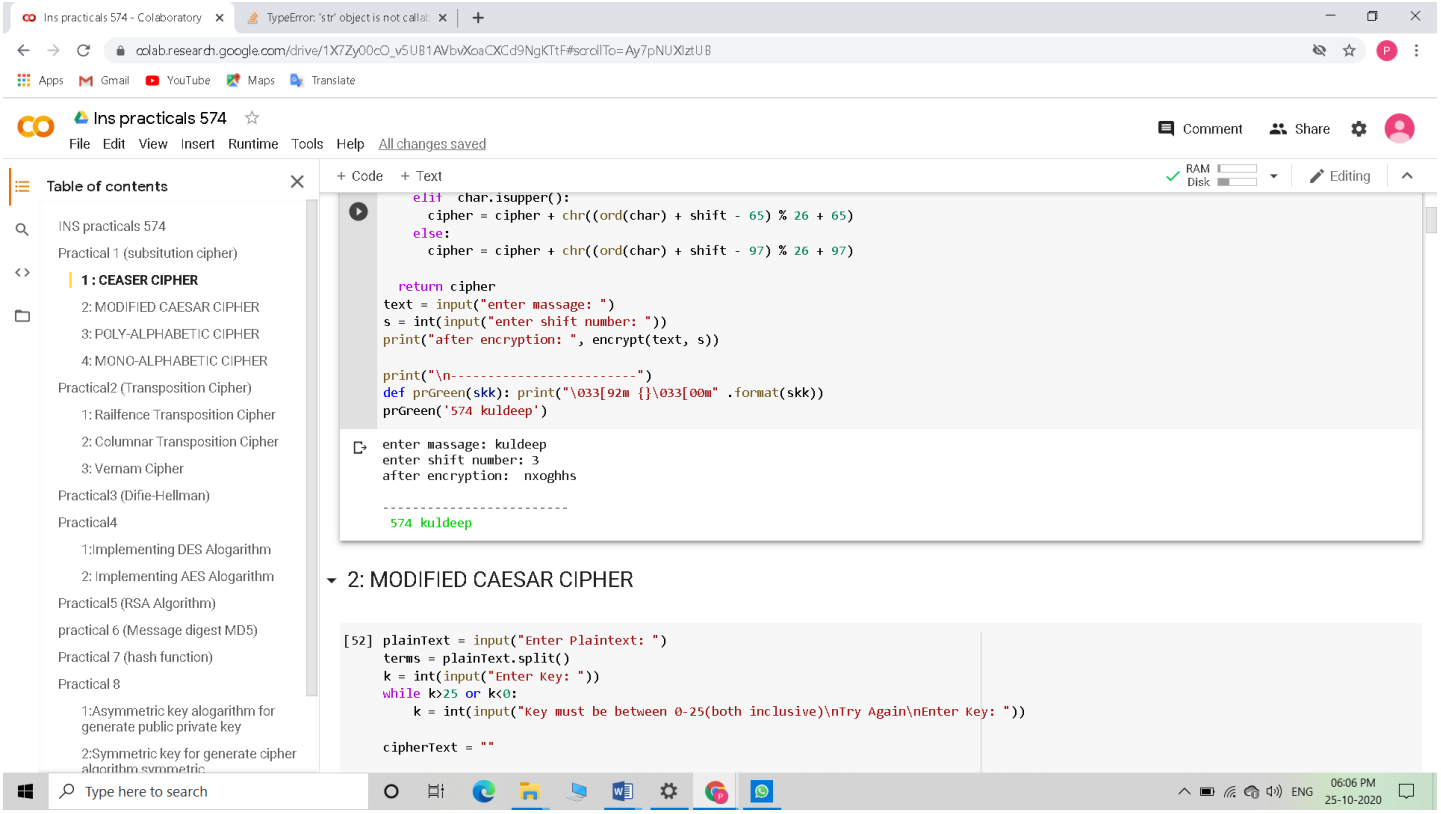
  For decryption phase with shift n

$$D_n(x) = (x-n) \bmod 26$$

Code:



Output :



Colab link :

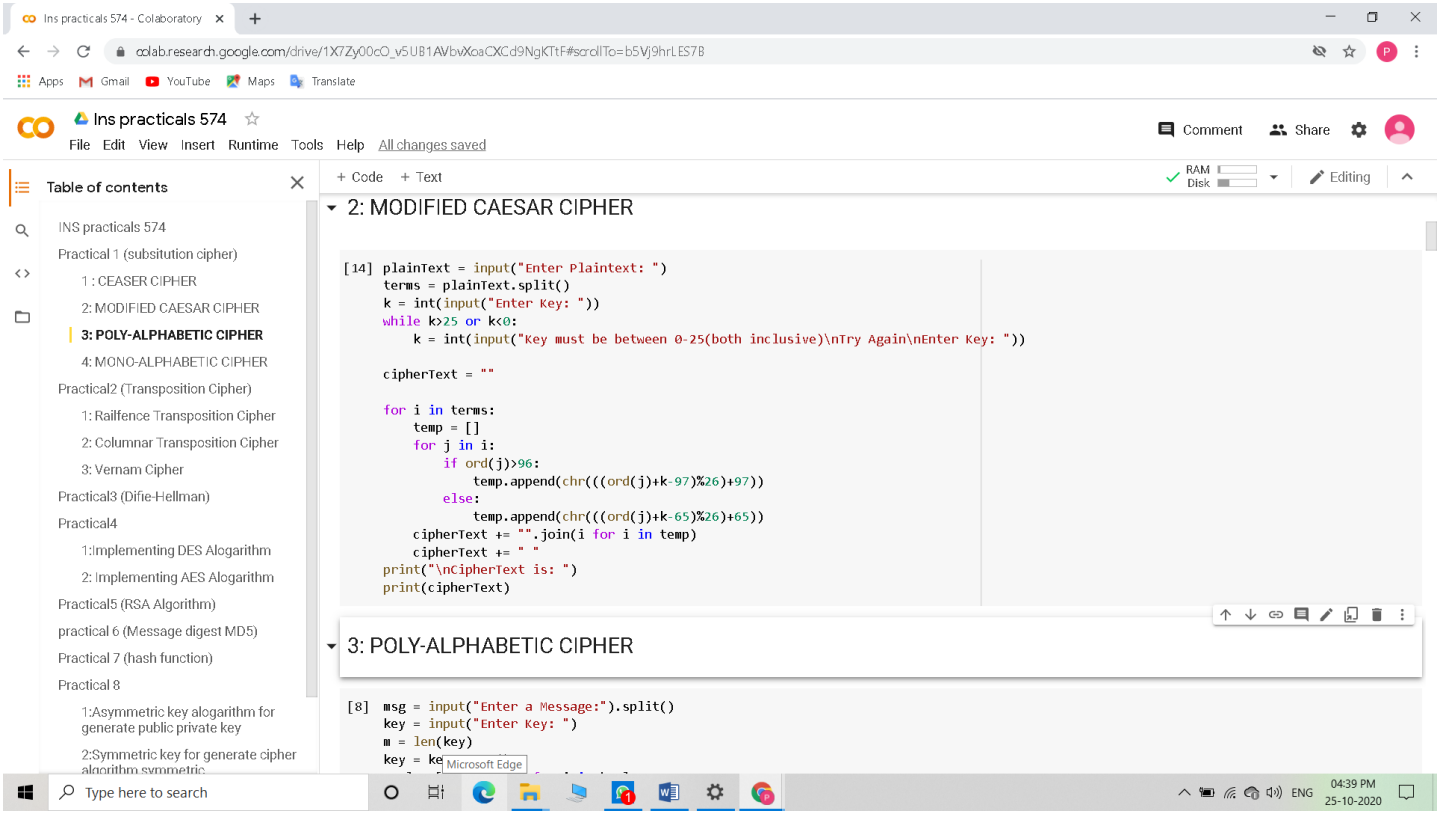
- https://colab.research.google.com/drive/1X7Zy00cO_v5UB1AVbvXoaCXCd9NgKTtF#scrollTo=UvHqRt3Gm81N

2] MODIFIED CAESAR CIPHER

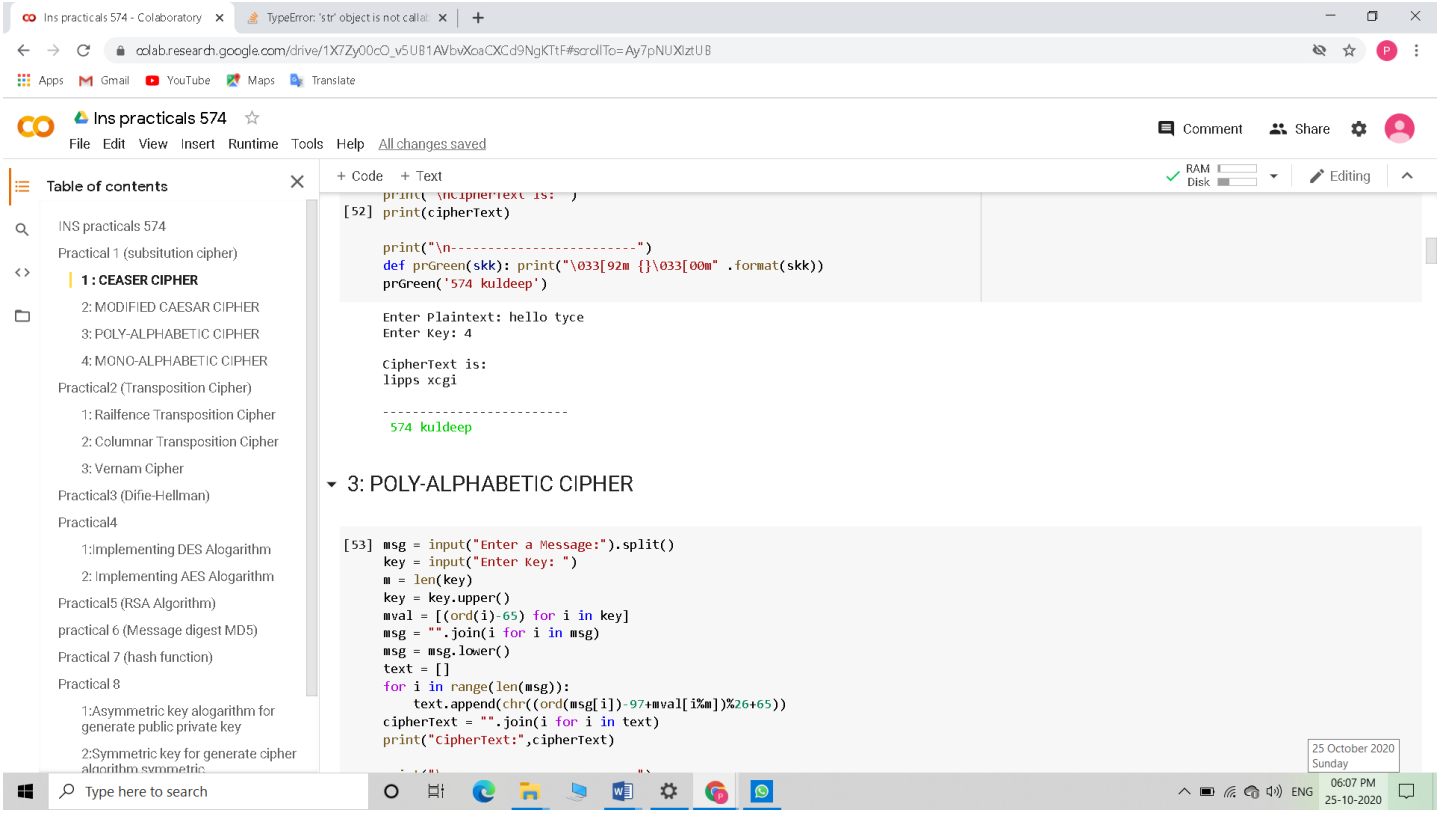
Theory:

- Modified Caesar cipher is an extension to Caesar cipher.
- **Modified Caesar cipher** algorithm to encrypt a message proposed algorithm requires plaintext and **encryption** key
- Caesar cipher is not good because it can be analysed be any attacker easily, so new concept was implemented to complicate the Caesar cipher and increase the complexity of the attacker to decode it.
- In **Modified Caesar cipher** each alphabet of plain text is may not necessarily replaced by key bits down the order instead the value of key is incremented and then it is replaced with new key value

Code:



Output :



Colab link :

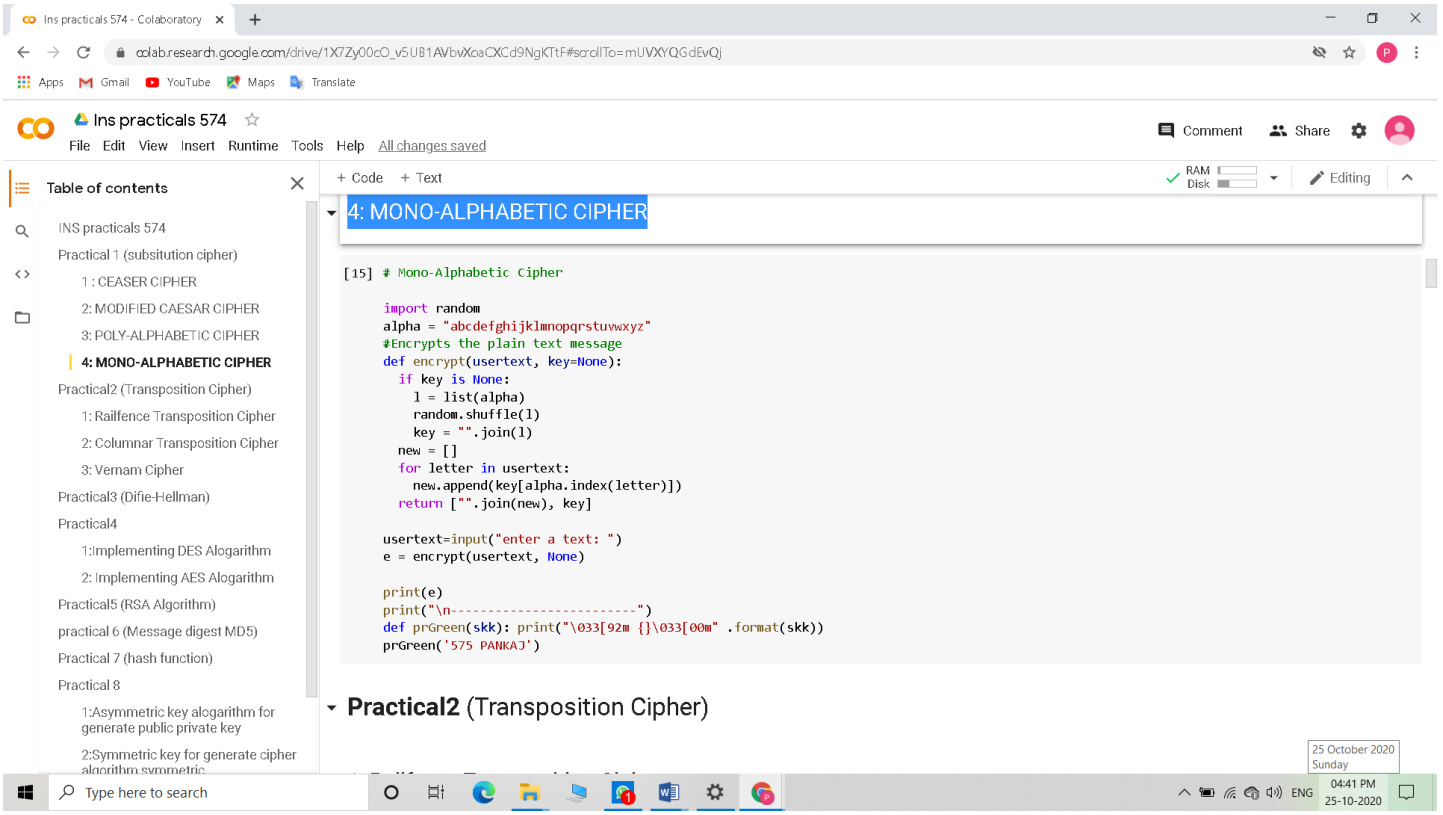
- https://colab.research.google.com/drive/1X7Zy00cO_v5UB1AVbvXoaCXCd9NgKTtF#scrollTo=_g2UyCRdDbJx

3] MONO – ALPHABETIC

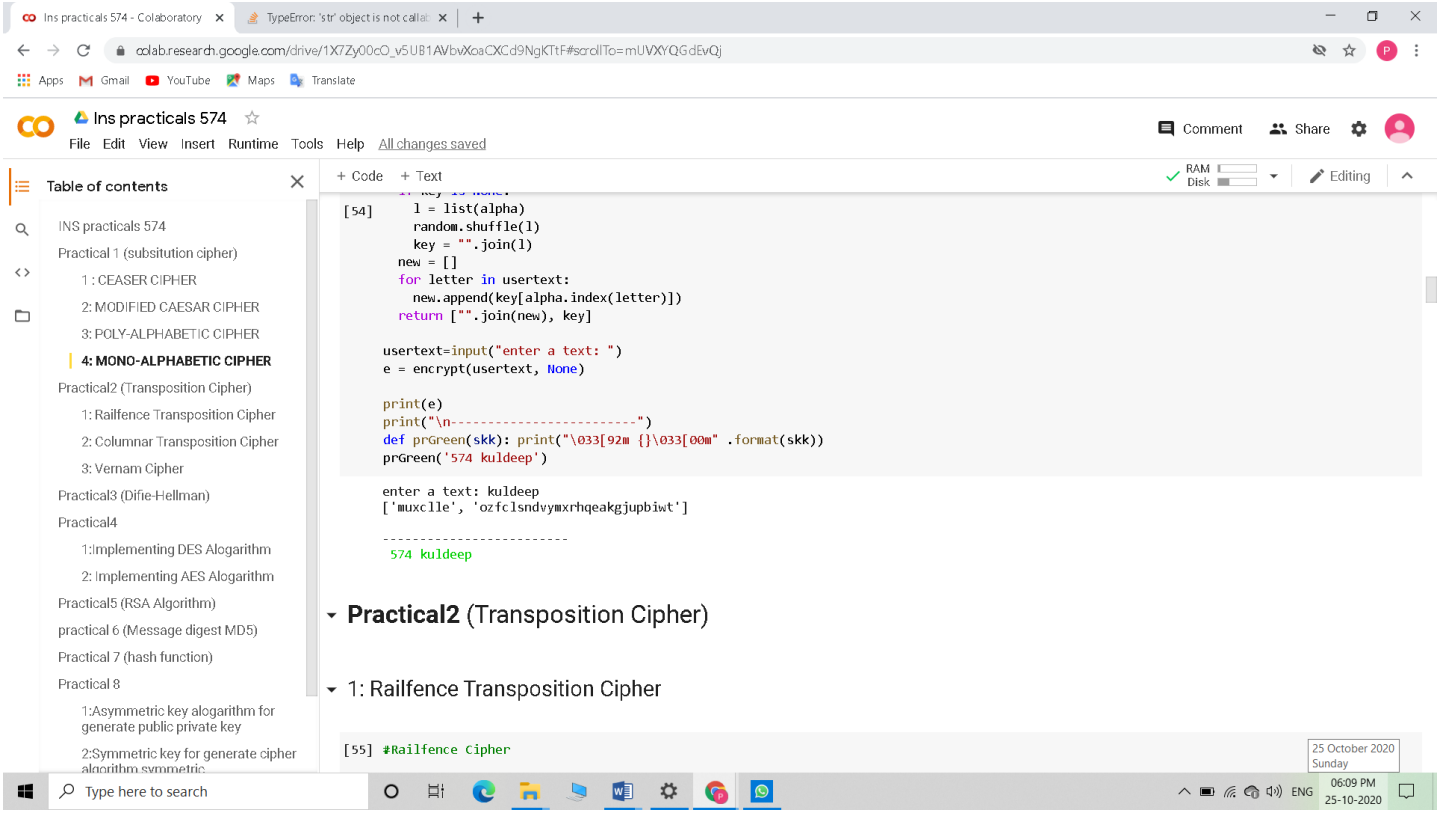
Theory:

- It is a **mono-alphabetic** cipher wherein each letter of the plaintext is substituted by another letter to form the ciphertext.
- The concept is to replace each **alphabet** by another **alphabet** which is 'shifted' by some fixed number between 0 and 25.
- For example, if ‘A’ is encrypted as ‘D’, for any number of occurrence in that plaintext, ‘A’ will always get encrypted to ‘D’.

Code:



Output:



Colab link:

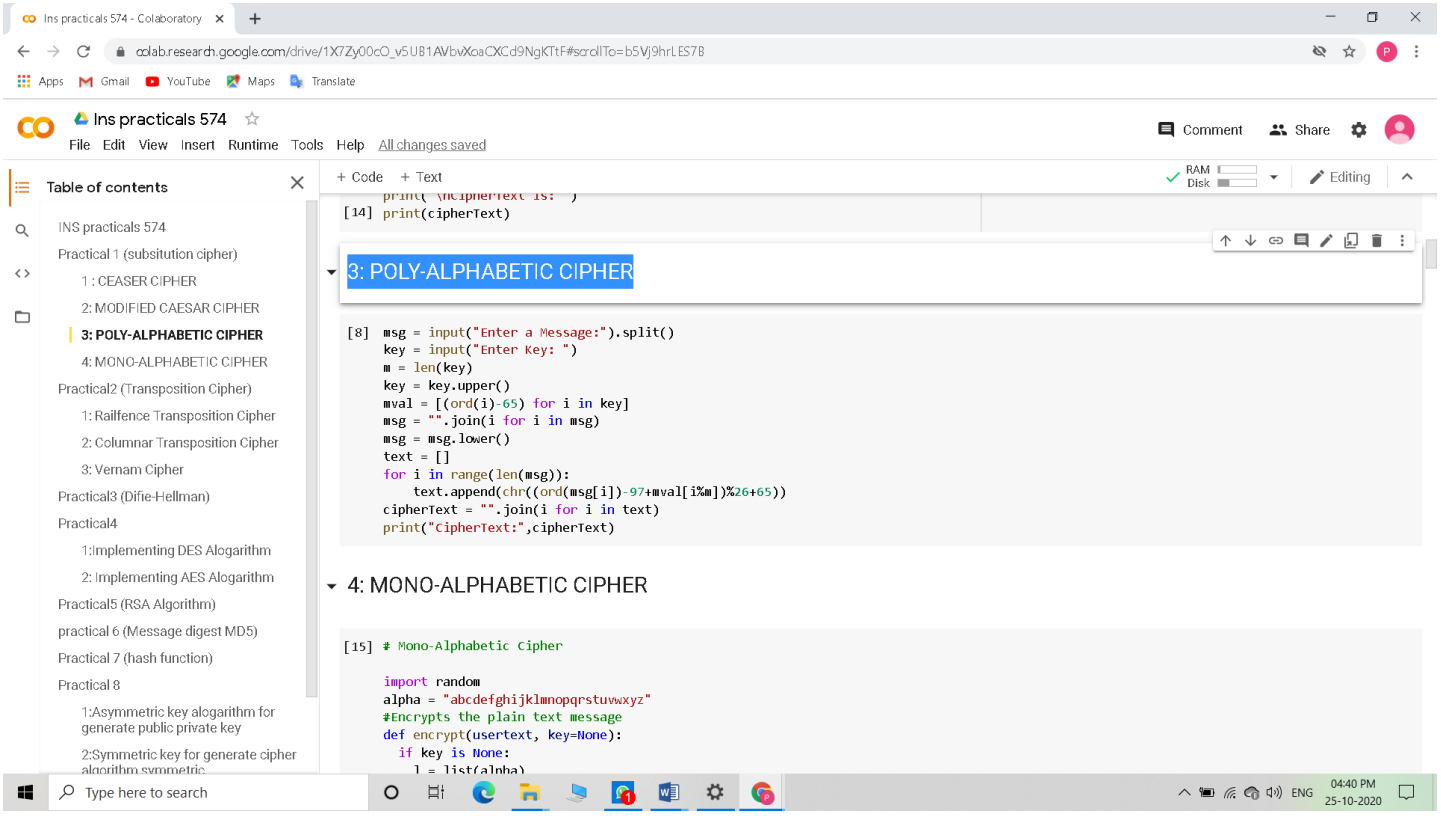
- https://colab.research.google.com/drive/1X7Zy00cO_v5UB1AVbvXoaCXCd9NgKTtF#scrollTo=mUVXYQGdEvQj

4] POLY – ALPHABETIC (VERNAM)

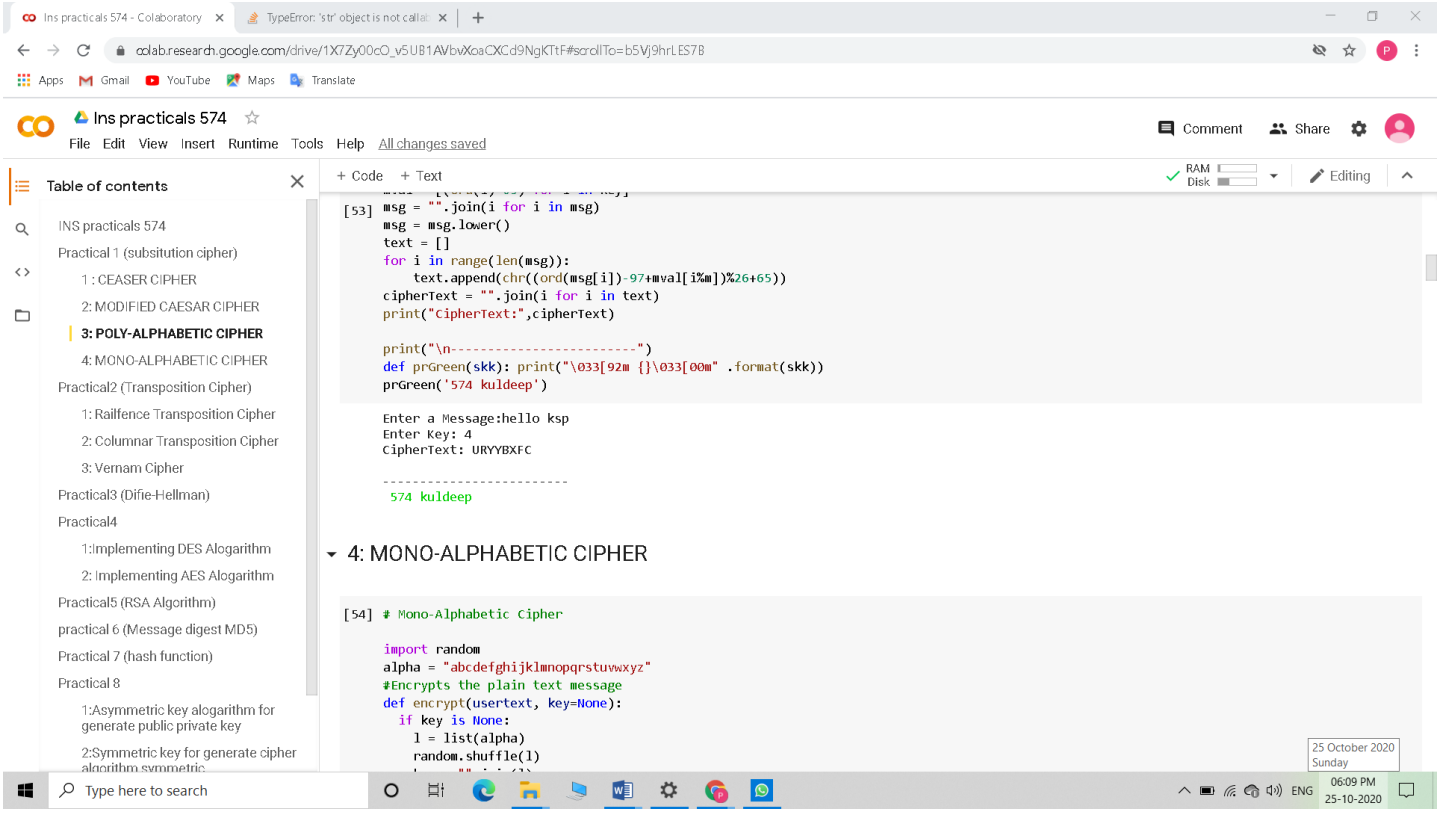
Theory:

- Polyalphabetic Cipher is a substitution cipher in which the cipher alphabet for the plain alphabet may be different at different places during the encryption process.
- The next two examples, **playfair and Vigenere Cipher are polyalphabetic ciphers**.

Code:



Output:



Colab link :

- https://colab.research.google.com/drive/1X7Zy00cO_v5UB1AVbvXoaCXCd9NgKTtF#scrollTo=b5Vj9hrLES7B

Practical 2

Aim : implementin the traposition cipher.

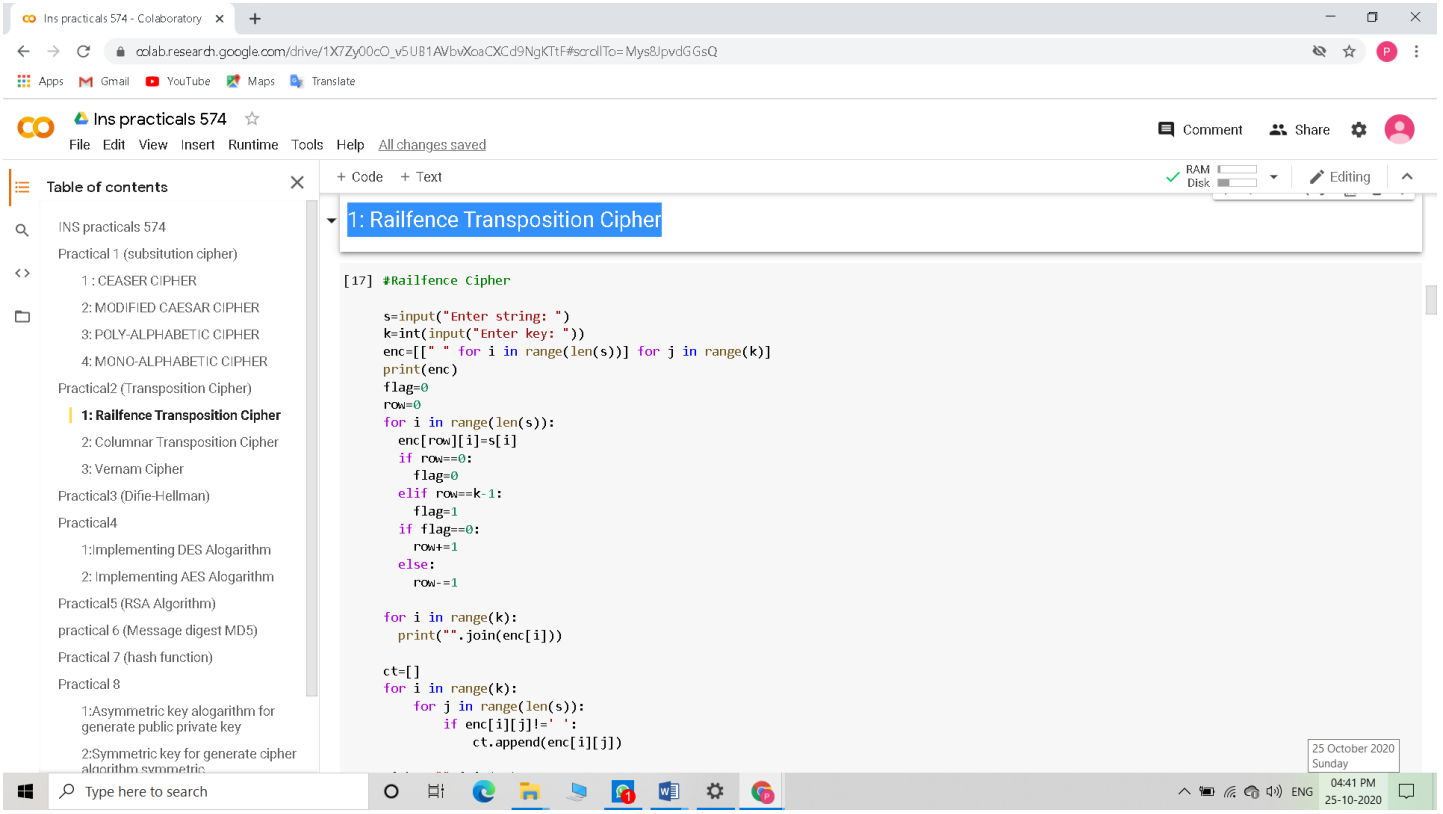
1] RAILFENCE TRAPOSITION CIPHER

Theory:

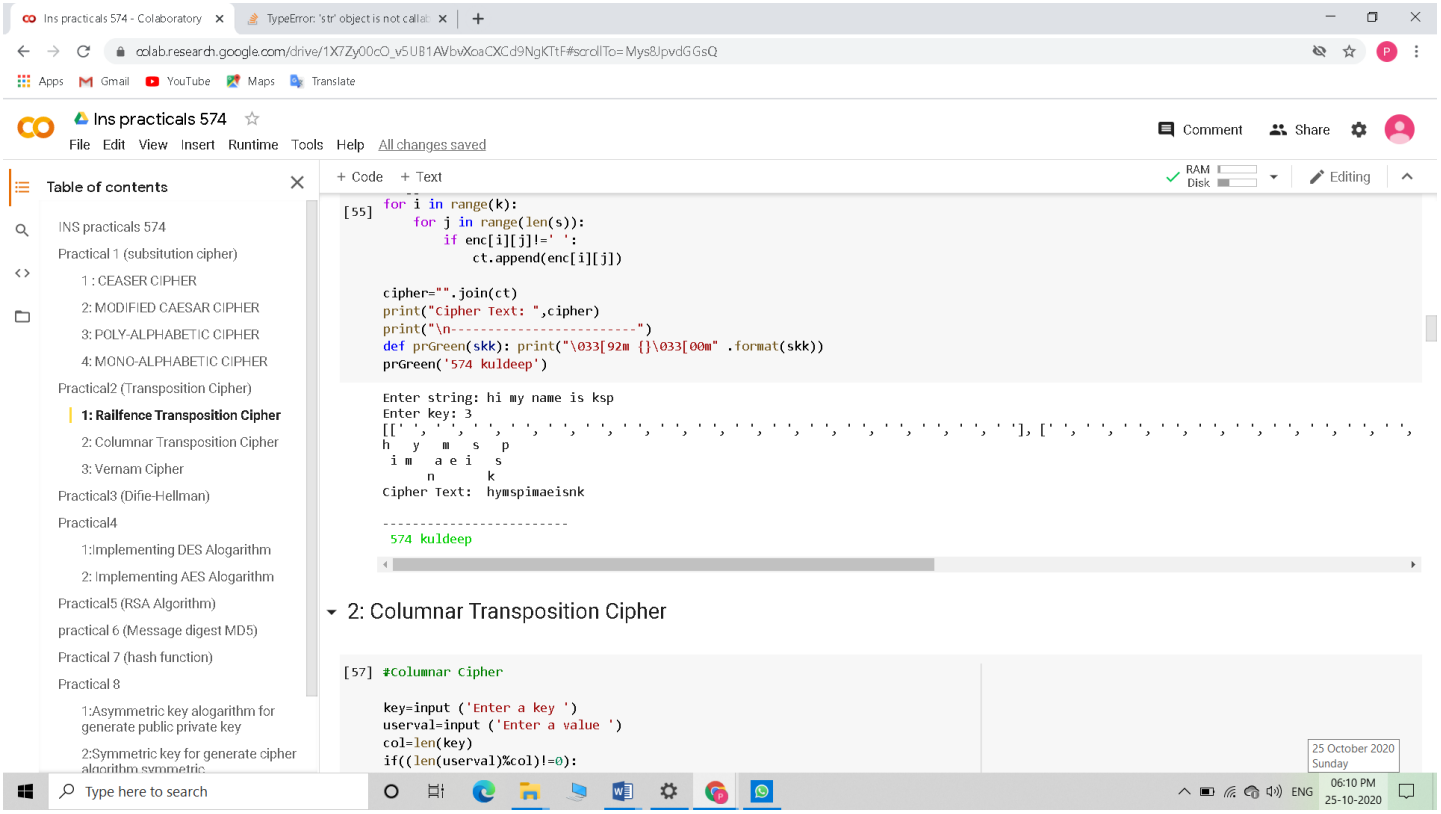
- The **rail fence cipher** (sometimes called zigzag **cipher**) is a **transposition cipher**
- It jumbles up the order of the letters of a message using a basic algorithm.
- The **rail fence cipher** works by writing your message on alternate lines across the page, and then reading off each line in turn.
- For example, let's consider the **plaintext** "This is a secret message".

Plaintext	T	H	I	S	I	S	A	S	E	C	R	E	T	M	E	S	S	A	G	E
Rail Fence	T		I		I		A		E		R		T		E		S		G	
Encoding		H		S		S		S		C		E		M		S		A		E

Code:



Output:



Colab link:

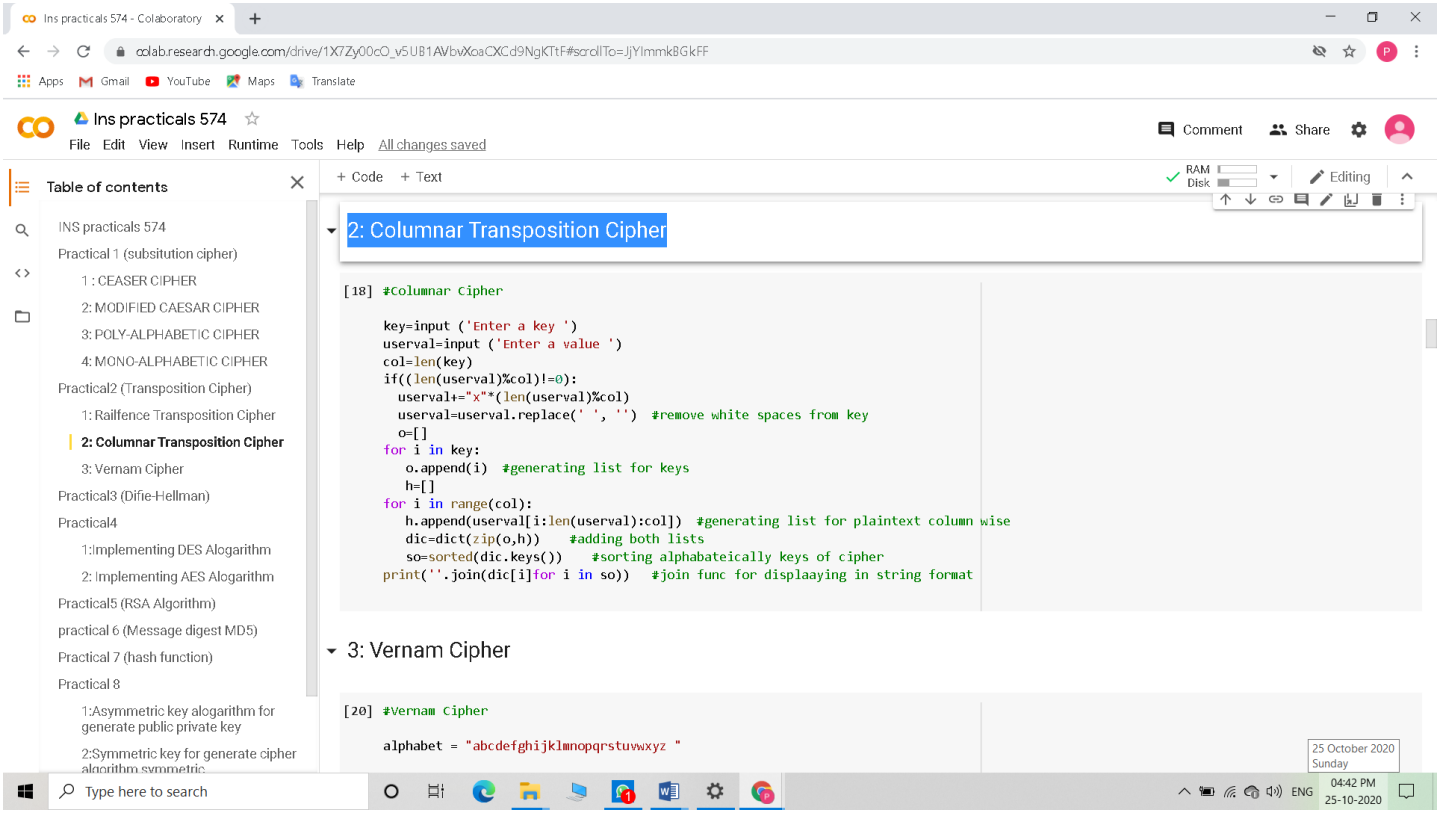
- https://colab.research.google.com/drive/1X7Zy00cO_v5UB1AVbvXoaCXCd9NgKTtF#scrollTo=Mys8JpvdGGsQ

2] SIMPLE COLUMNER TRANSPOSITION

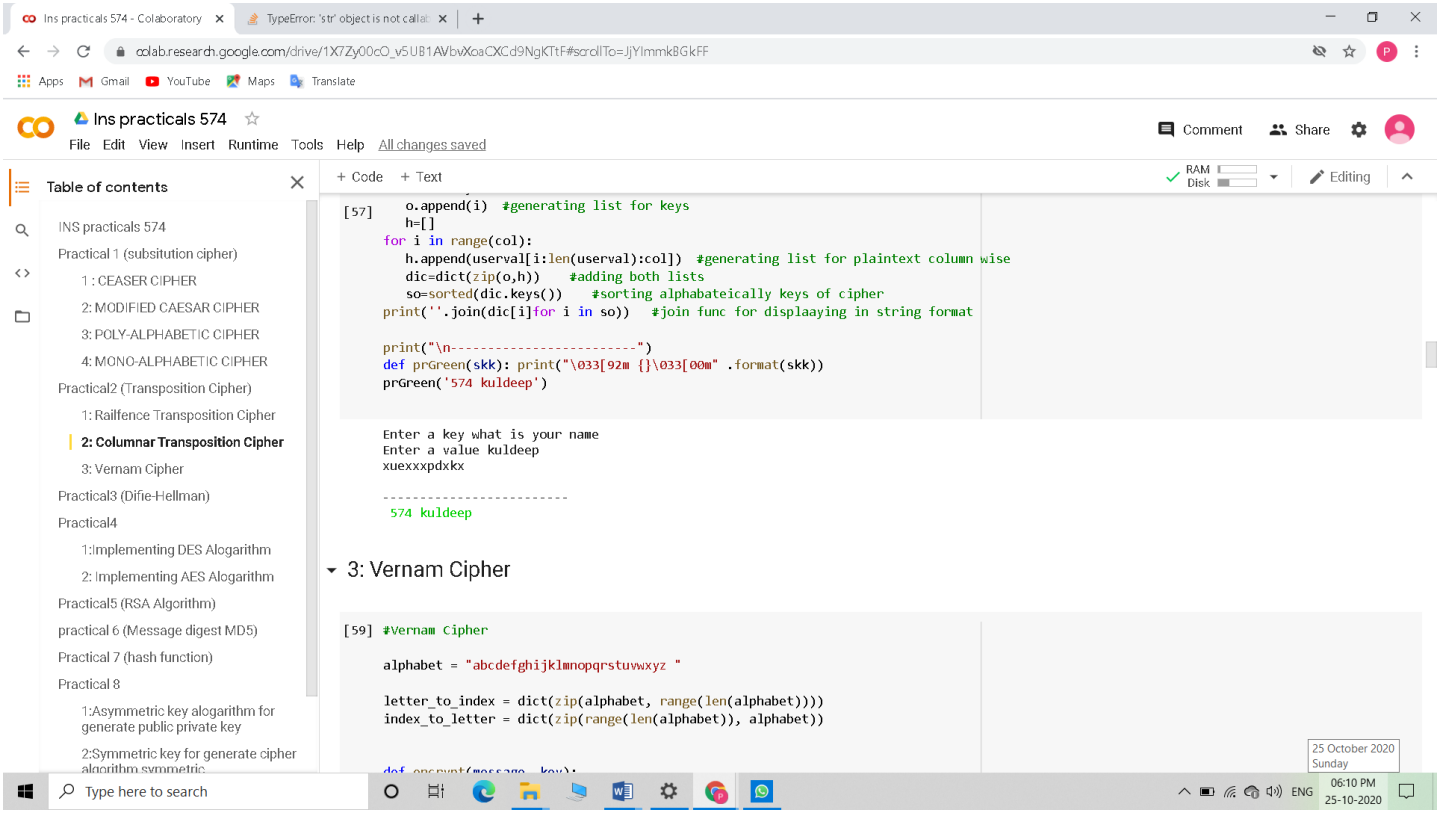
Theory:

- **Columnar Transposition** involves writing the plaintext out in rows.
- And then reading the ciphertext off in columns.
- it is the Route Cipher where the route is to read down each column in order. For example, the plaintext "a **simple transposition**" with 5 columns looks like the grid below.

Code:



Output:



Colab link:

- https://colab.research.google.com/drive/1X7Zy00cO_v5UB1AVbvXoaCXCd9NgKTtF#scrollTo=JjYImmkBGkFF

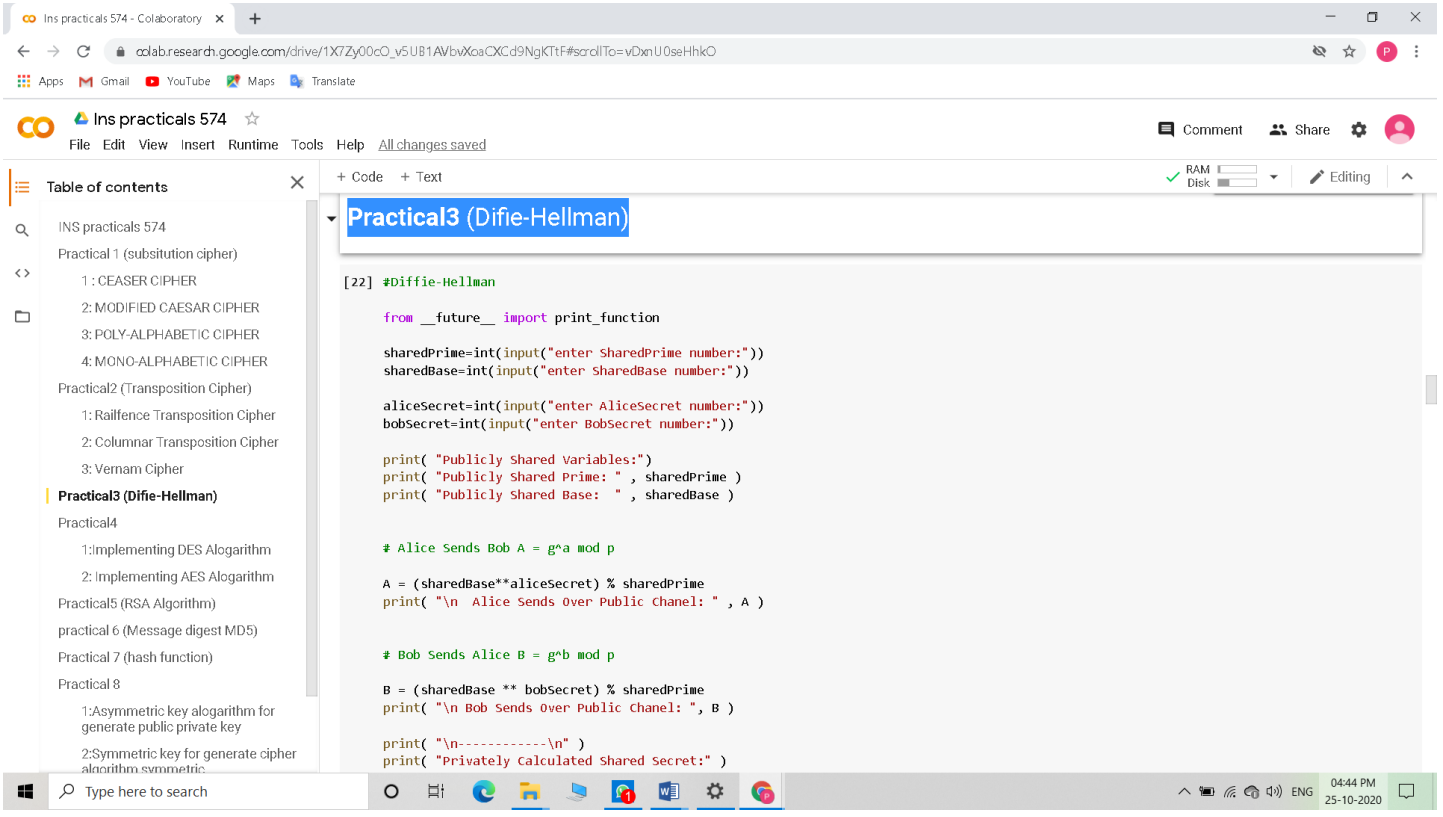
Practical 3

Aim: implementing the dilfie hellman key exchange algorithm.

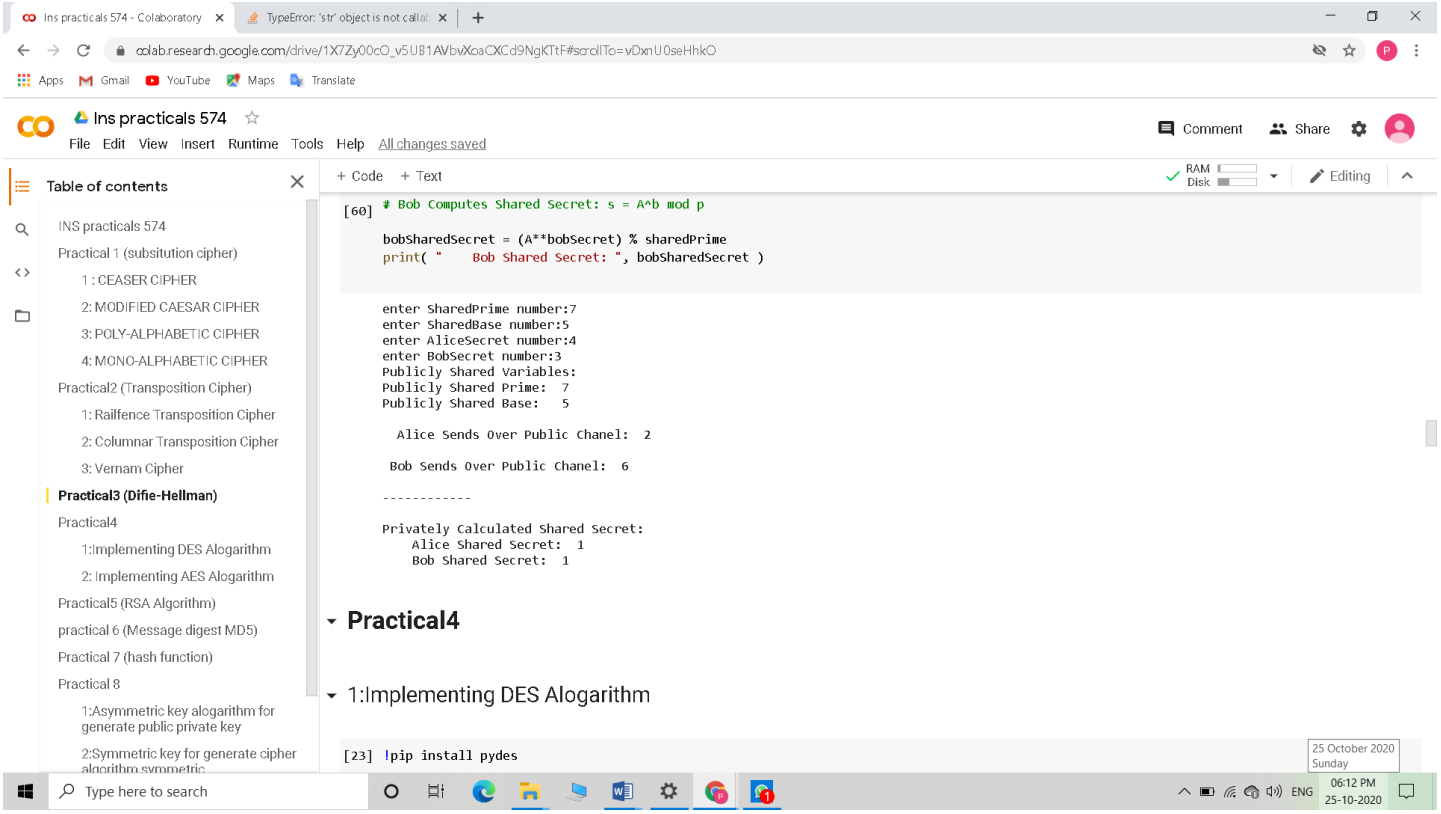
Theory:

- **Diffie Hellman** (**DH**) **key exchange algorithm** is a method for securely exchanging cryptographic **keys** over a public communications channel.
- **Keys** are not actually **exchanged** – they are jointly derived.
- It is named after their inventors Whitfield **Diffie** and Martin **Hellman**.
- The Diffie-Hellman algorithm is being used to establish a shared secret that can be used for secret communications
- while exchanging data over a public network using the elliptic curve to generate points and get the secret key using the parameters.

Code:



Output:



Colab link :

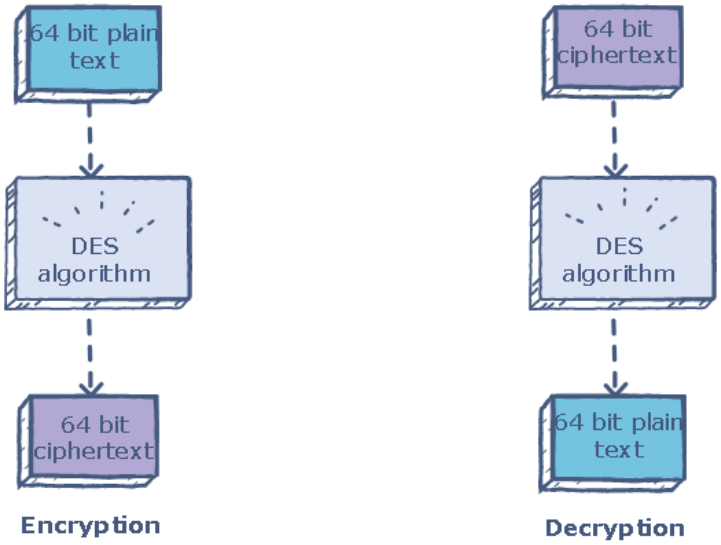
- https://colab.research.google.com/drive/1X7Zy00cO_v5UB1AVbvXoaCXCd9NgKTtF#scrollTo=vDxnU0seHhkO

Practical 4

Aim: 1] Implementing the DES alogarithm.

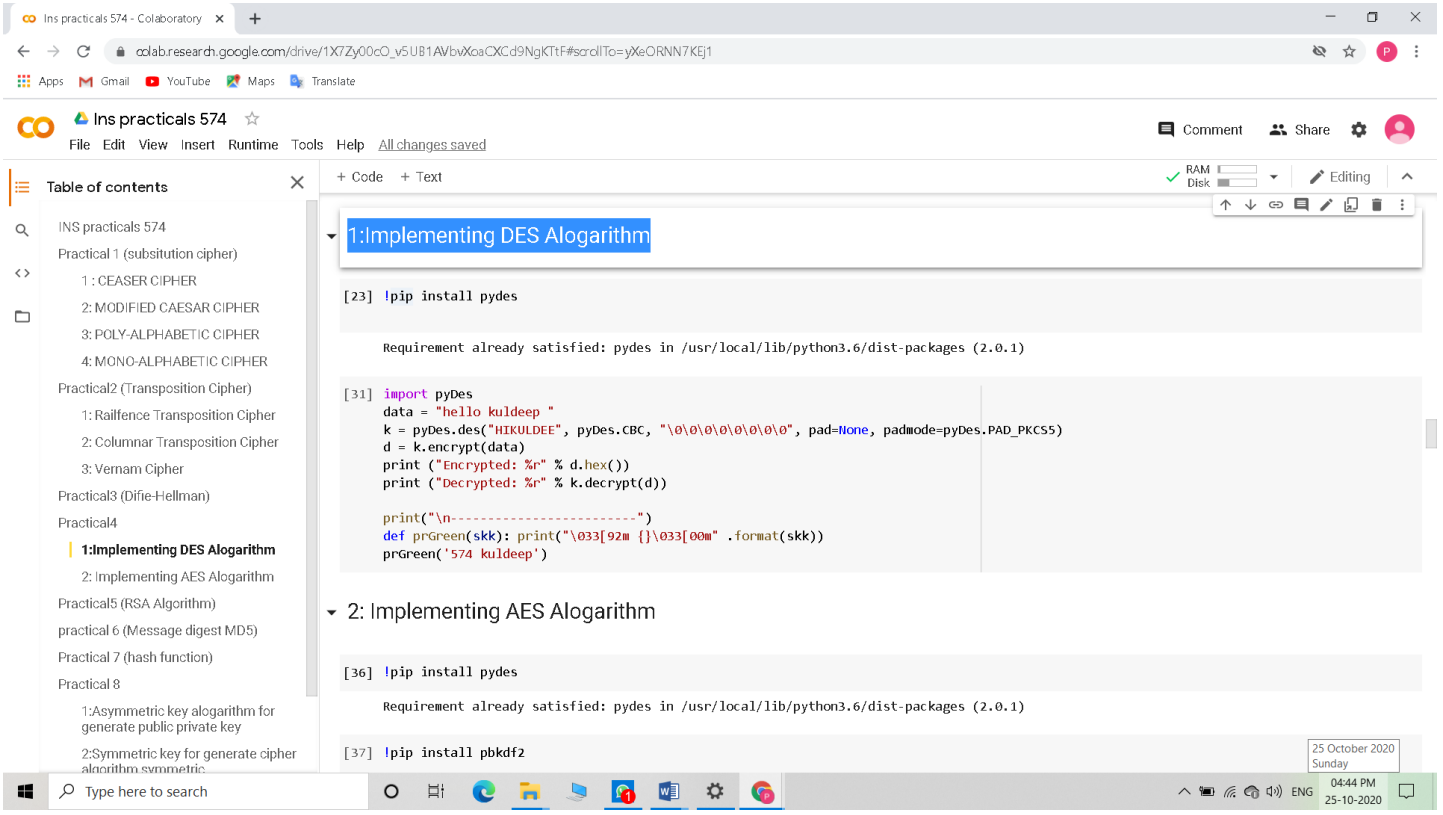
Theory:

- **Data Encryption Standard (DES)** is a block cipher algorithm that
- **It** takes plain text in blocks of 64 bits and converts them to ciphertext using keys of 48 bits.
- It is a symmetric key algorithm, which means that the same key is used for encrypting and decrypting data.

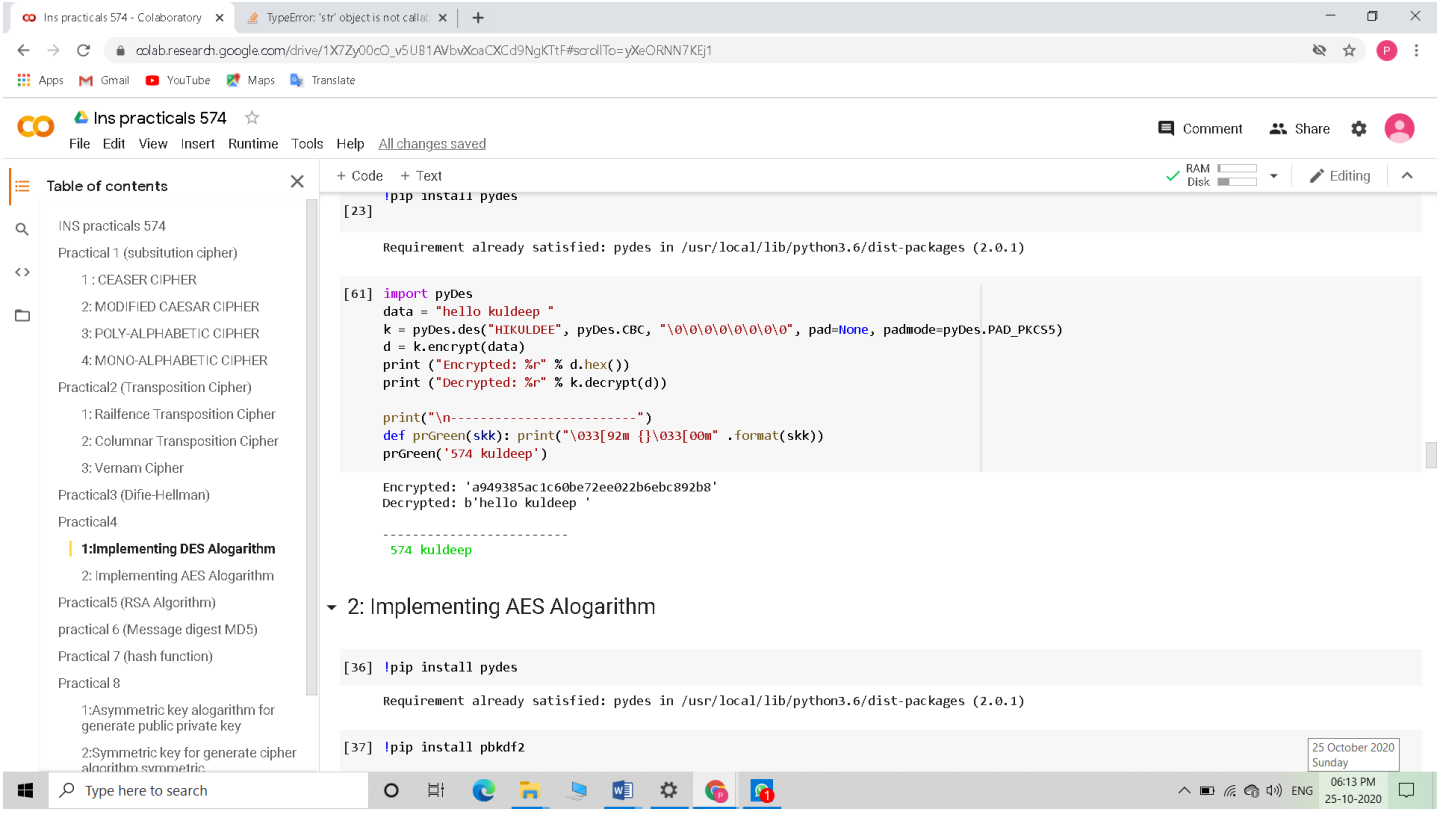


Encryption and decryption using the DES algorithm.

Code:



Output :



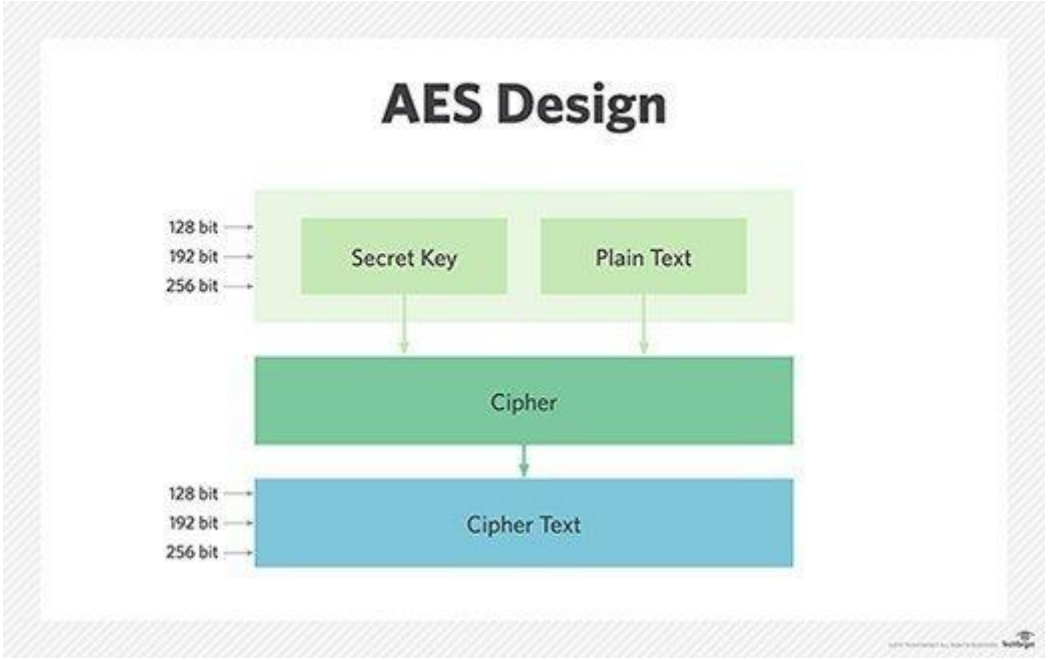
Colab link :

- https://colab.research.google.com/drive/1X7Zy00cO_v5UB1AVbvXoaCXCd9NgKTtF#scrollTo=yXeORNN7KEj1

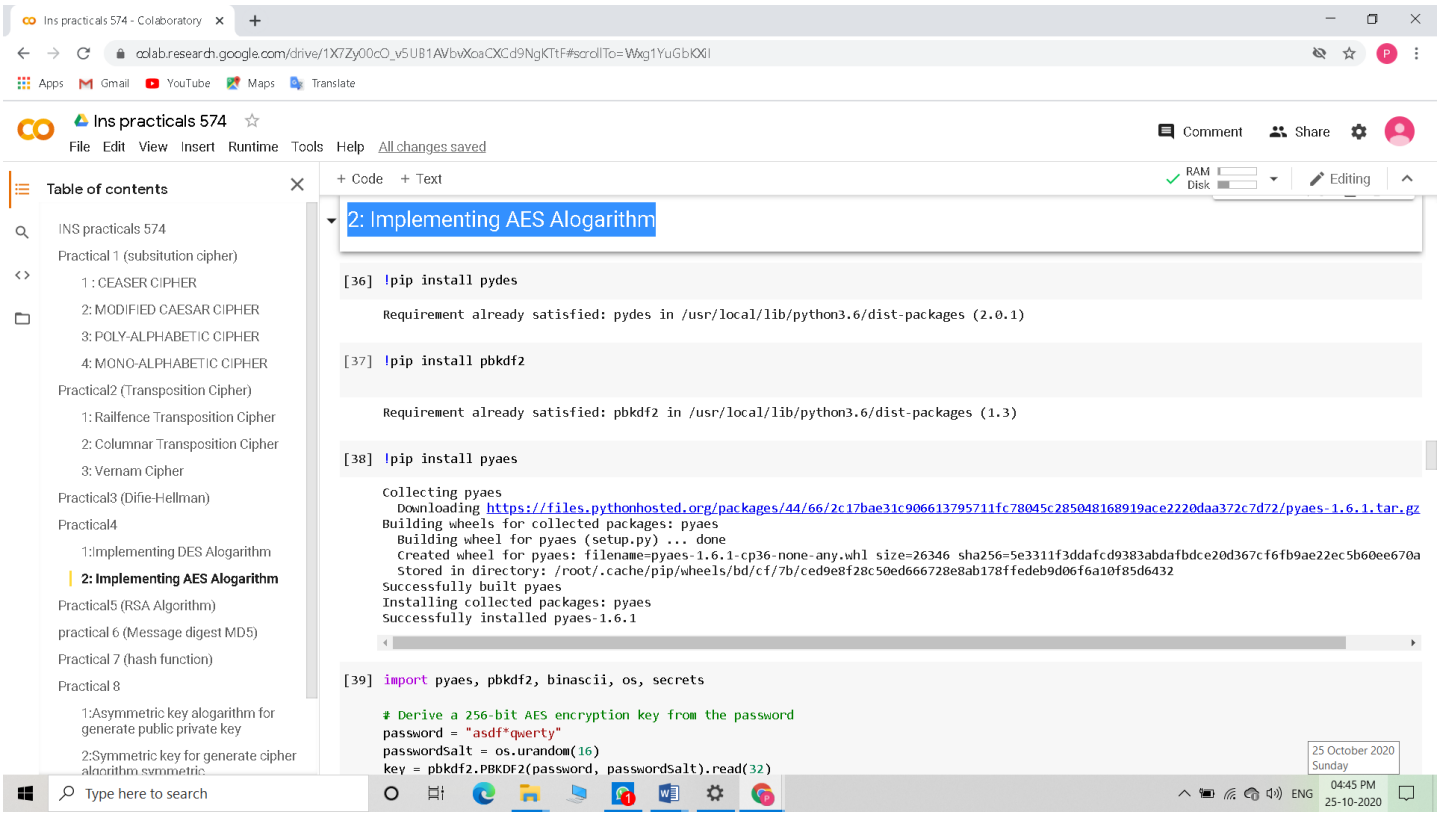
2] implementing the AES alogarithm

Theory:

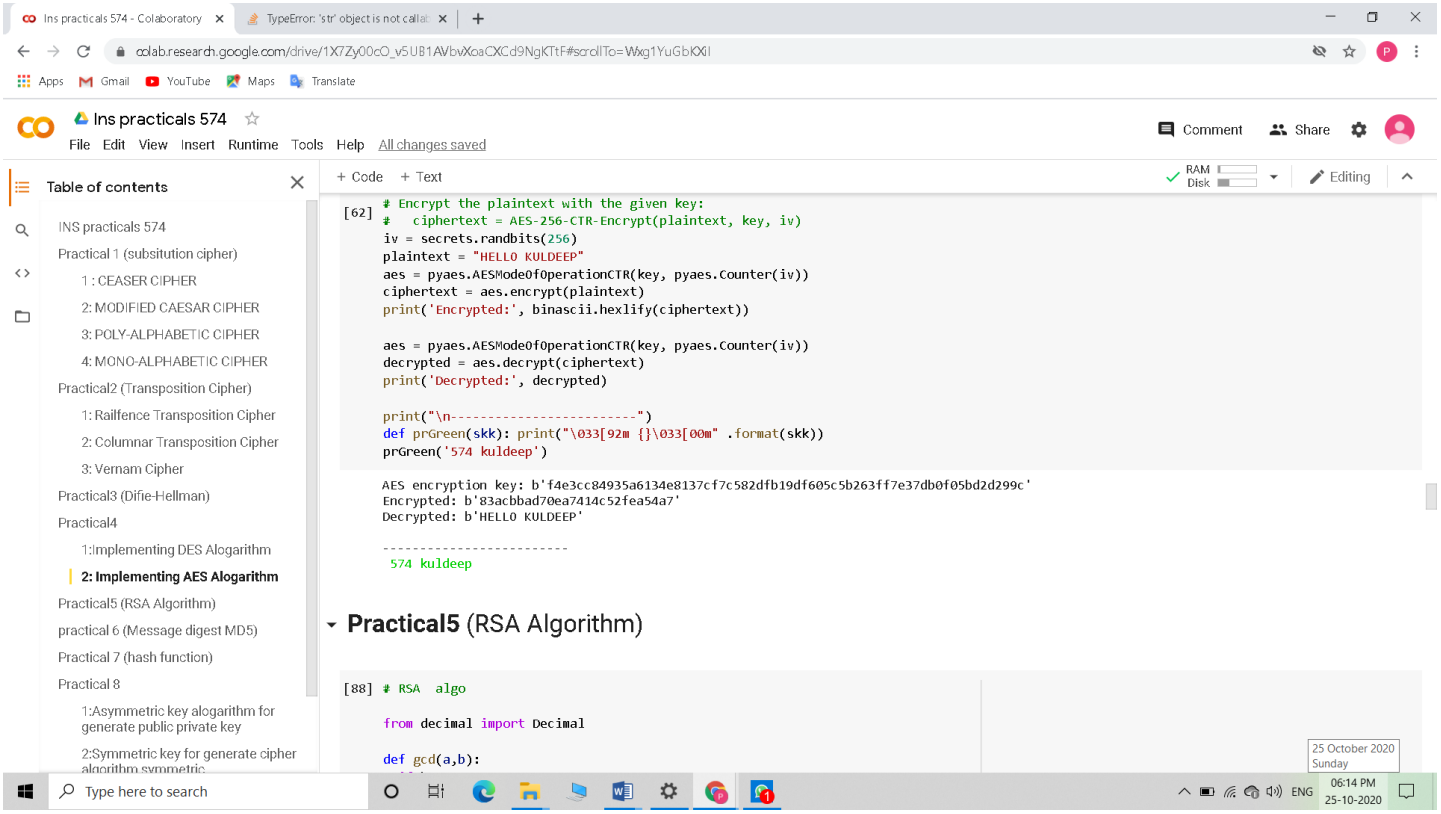
- AES is an iterative rather than Feistel cipher.
- It is based on ‘substitution–permutation network’.
- It comprises of a series of linked operations, some of which involve replacing inputs by specific outputs (substitutions) and others involve shuffling bits around (permutations).
- AES performs all its computations on bytes rather than bits. Hence, AES treats the 128 bits of a plaintext block as 16 bytes.



Code:



Output:



Colab link:

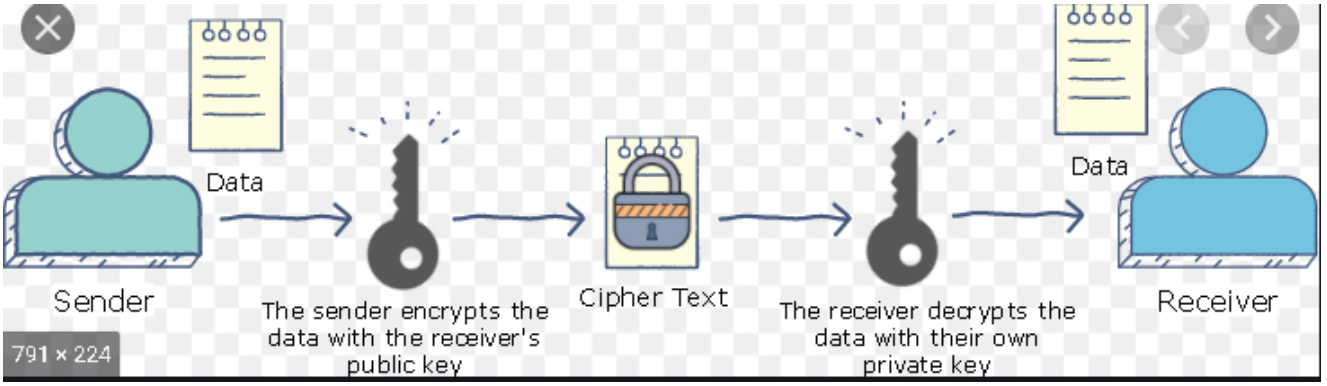
- https://colab.research.google.com/drive/1X7Zy00cO_v5UB1AVbvXoaCXCd9NgKTtF#scrollTo=Wxg1YuGbKXiI

Practical 5

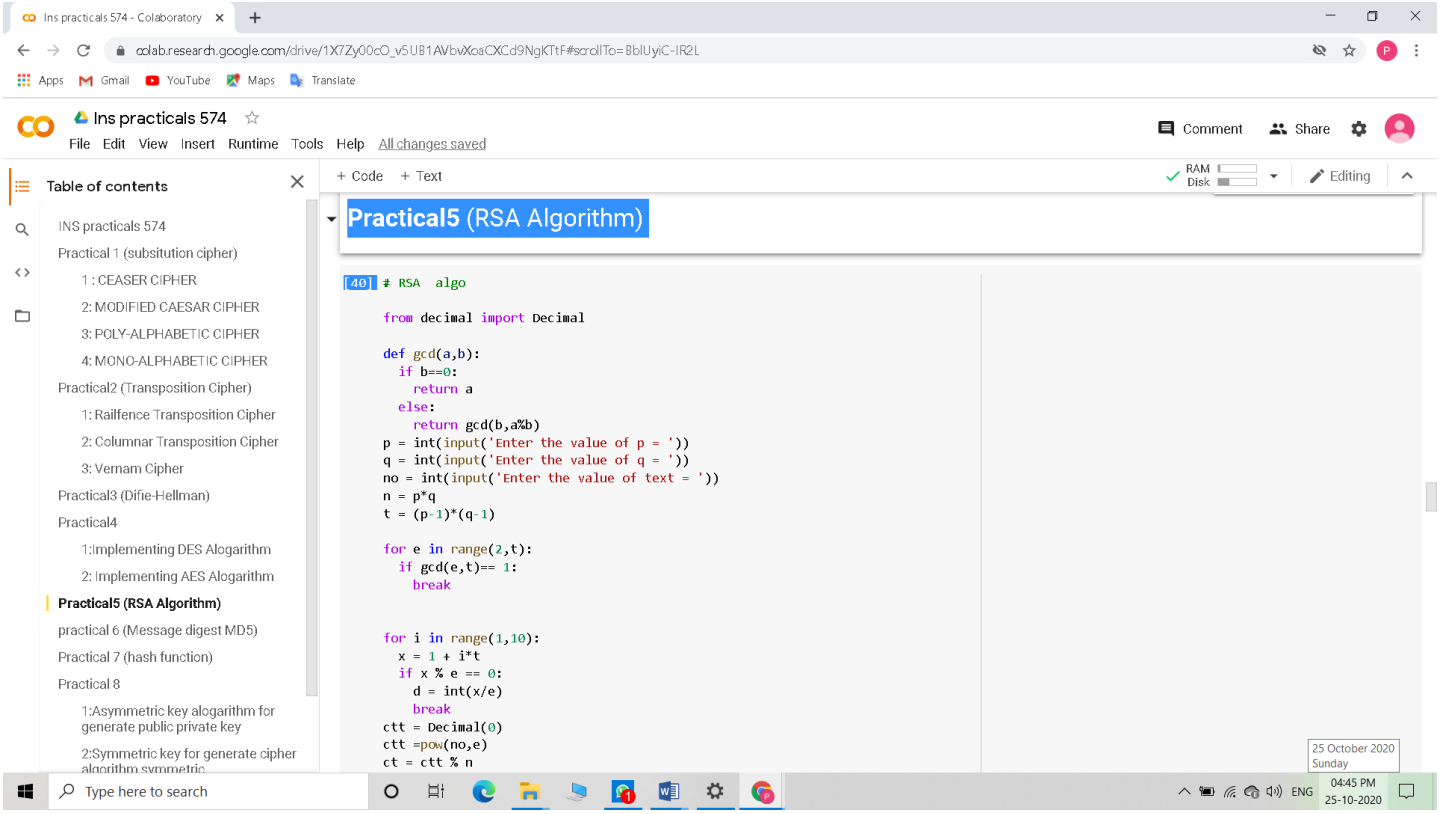
Aim : Implementing the RSA algorithm.

Theory :

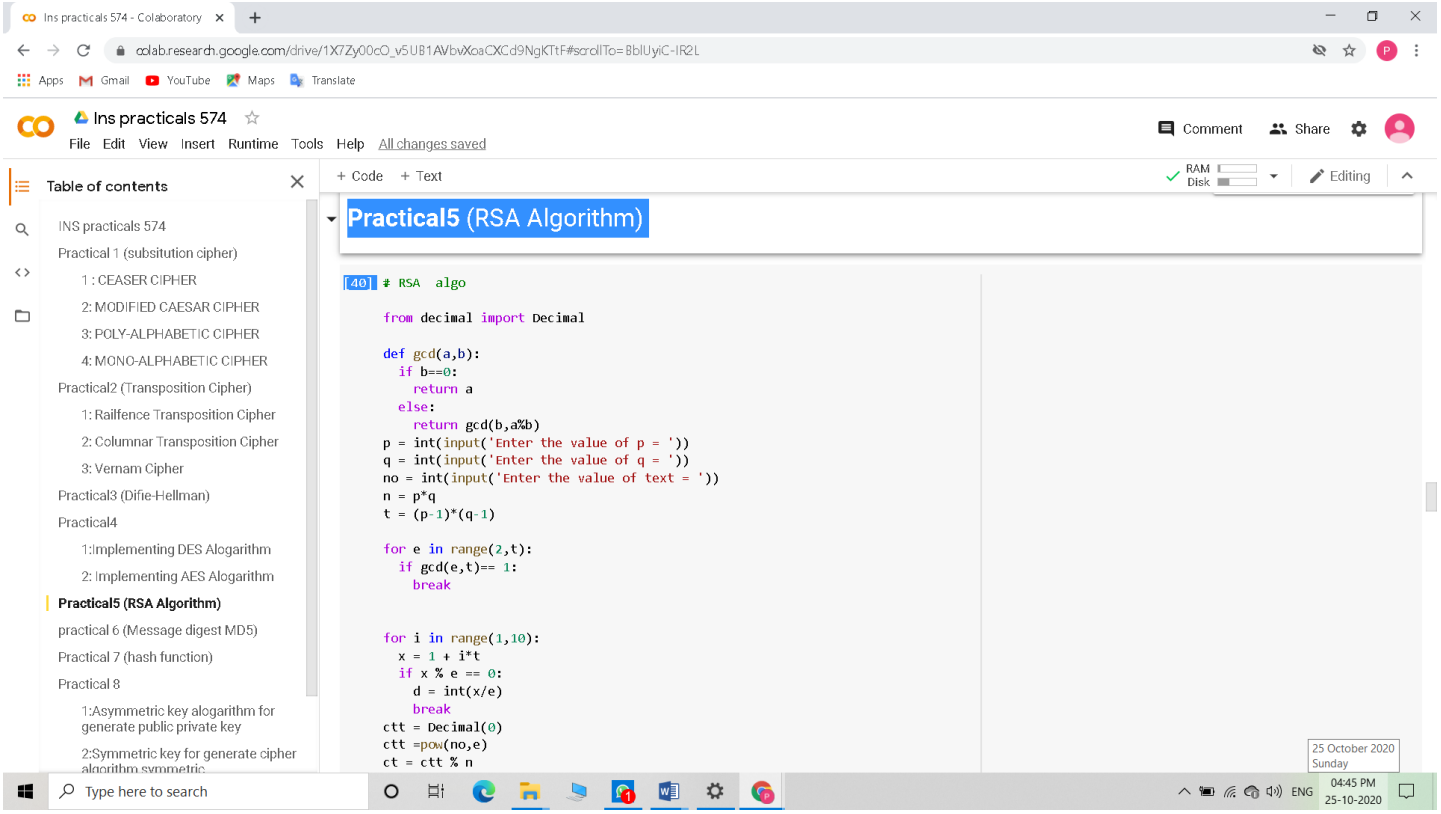
- **RSA** (Rivest–Shamir–Adleman) is an **algorithm** used by modern computers to encrypt and decrypt messages.
- It is an asymmetric cryptographic **algorithm**.
- Asymmetric means that there are two different keys.
- This is also called public key cryptography, because one of the keys can be given to anyone.



Code:



Output :



Colab link :

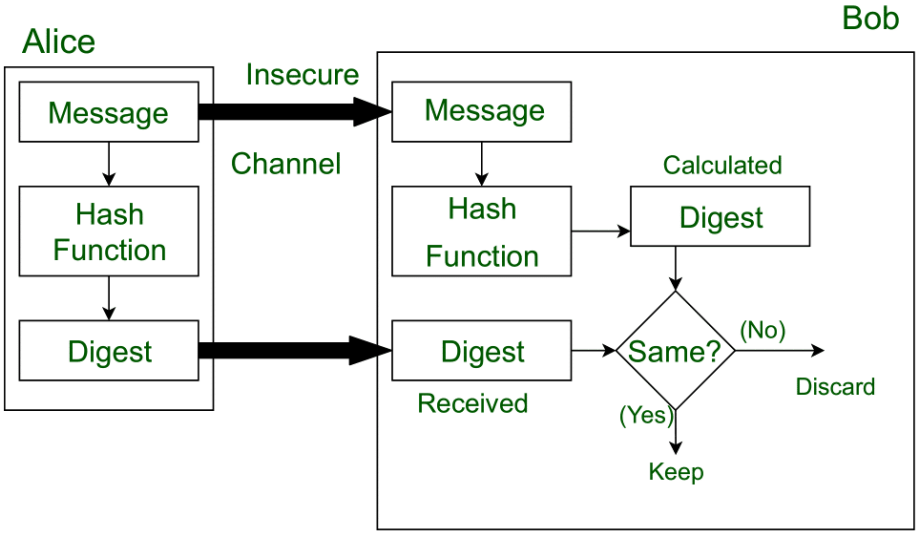
- https://colab.research.google.com/drive/1X7Zy00cO_v5UB1AVbvXoaCXCd9NgKTtF#scrollTo=BblUyiC-IR2L

Practical 6

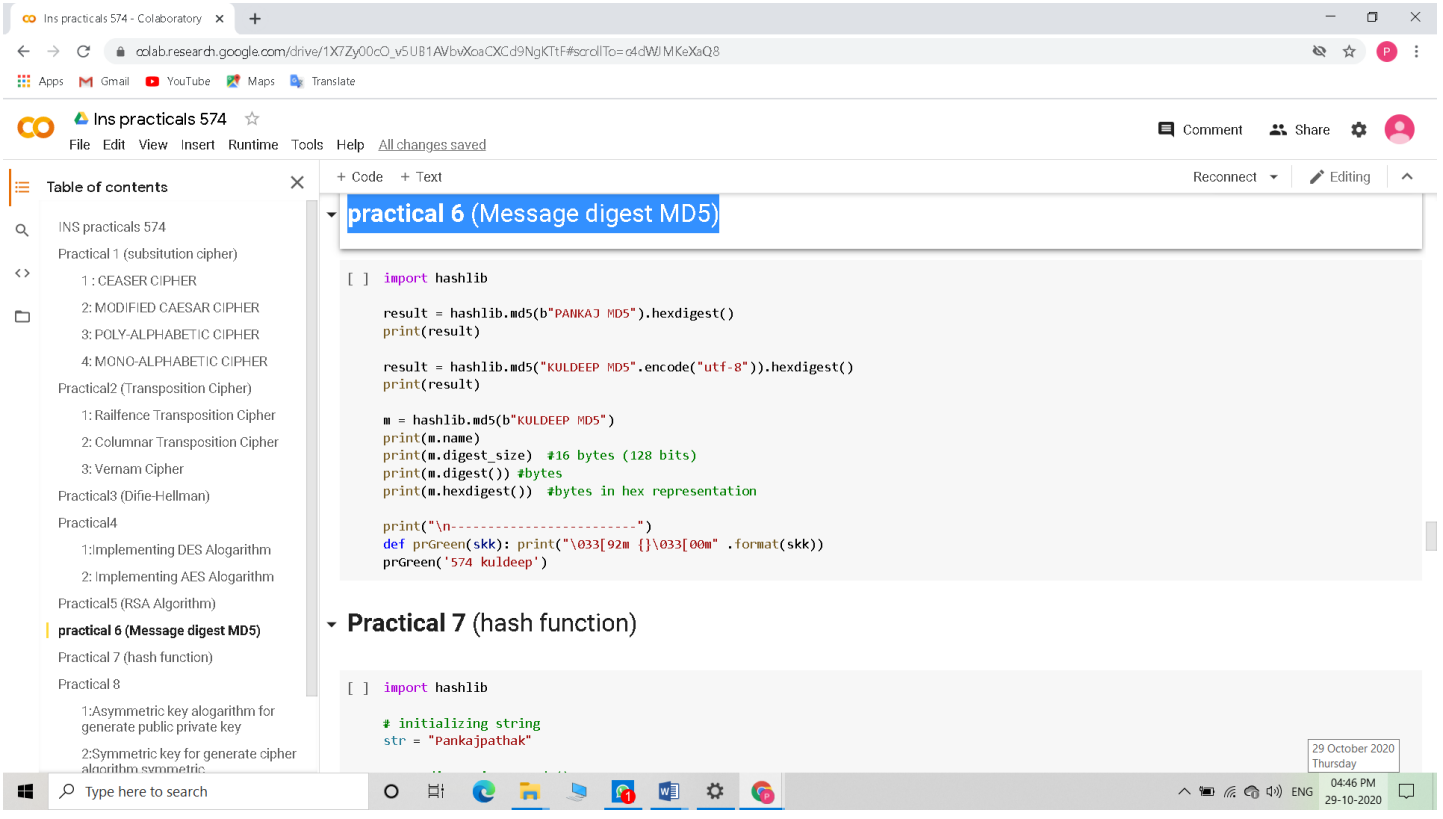
Aim : Message Digest (MD5)

Theory :

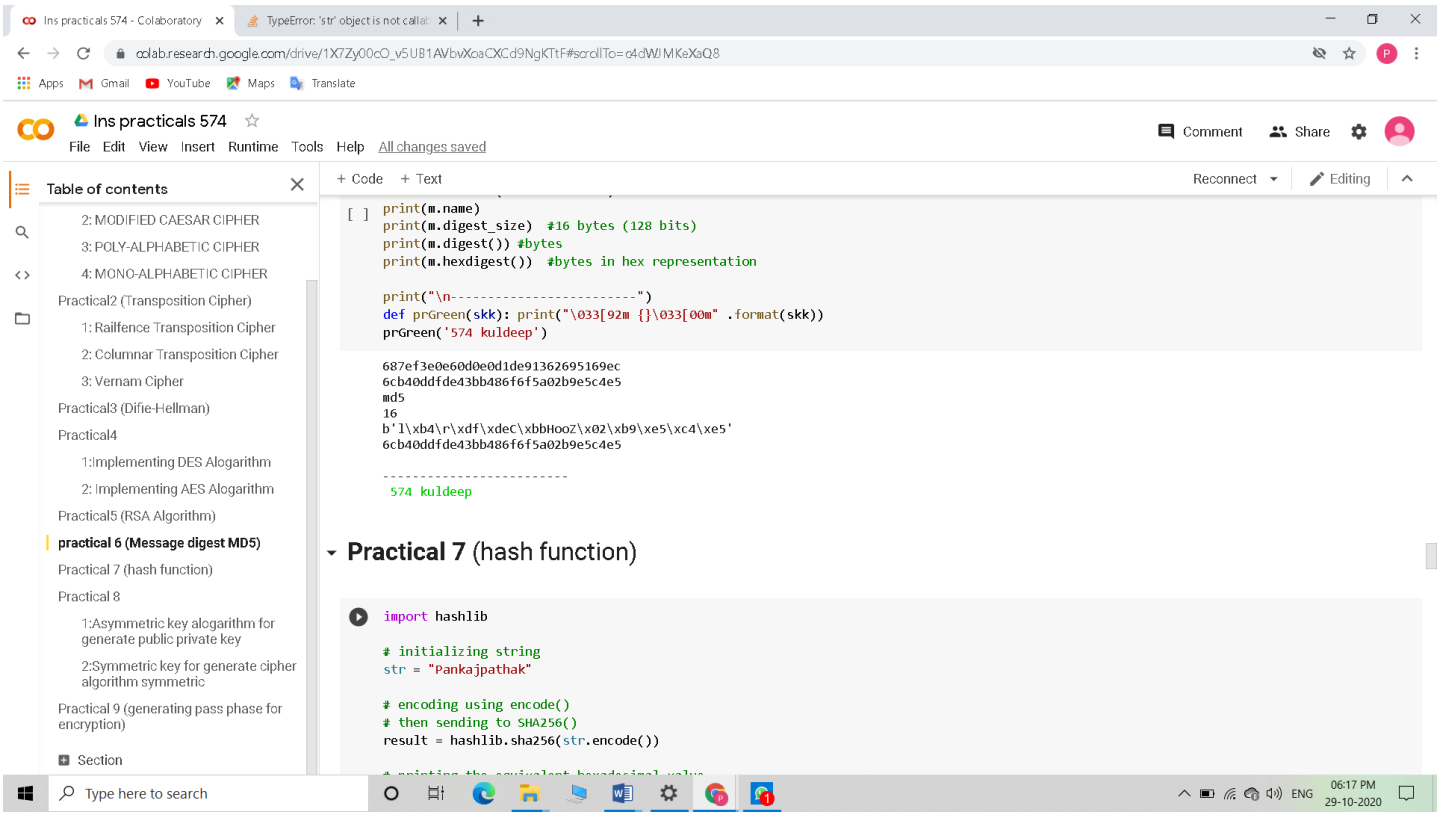
- **Message Digest** is used to ensure the integrity of a message transmitted over an insecure channel (where the content of the message can be changed).
- The message is passed through a Cryptographic hash function. This function creates a compressed image of the message called **Digest**.
- Alice sent a message and digest pair to Bob.
- To check the integrity of the message Bob runs the cryptographic hash function on the received message and gets a new digest.
- Now, Bob will compare the new digest and the digest sent by Alice. If, both are same then Bob is sure that the original message is not changed.



Code :



Output:



Colab link:

- https://colab.research.google.com/drive/1X7Zy00cO_v5UB1AVbvXoaCXCd9NgKTtF#scrollTo=c4dWJMKeXaQ8

Practical 7

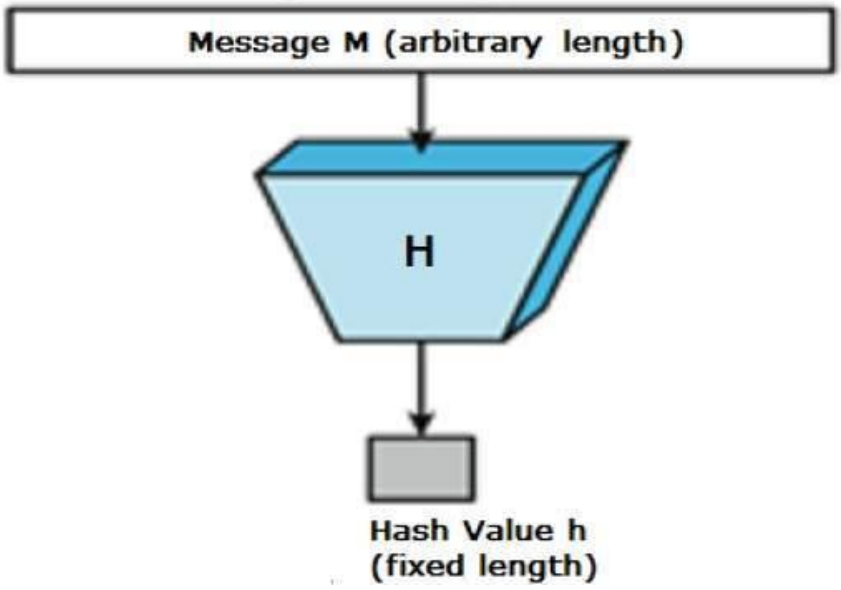
Aim : Hash function (HMAC SHA)

Theory :

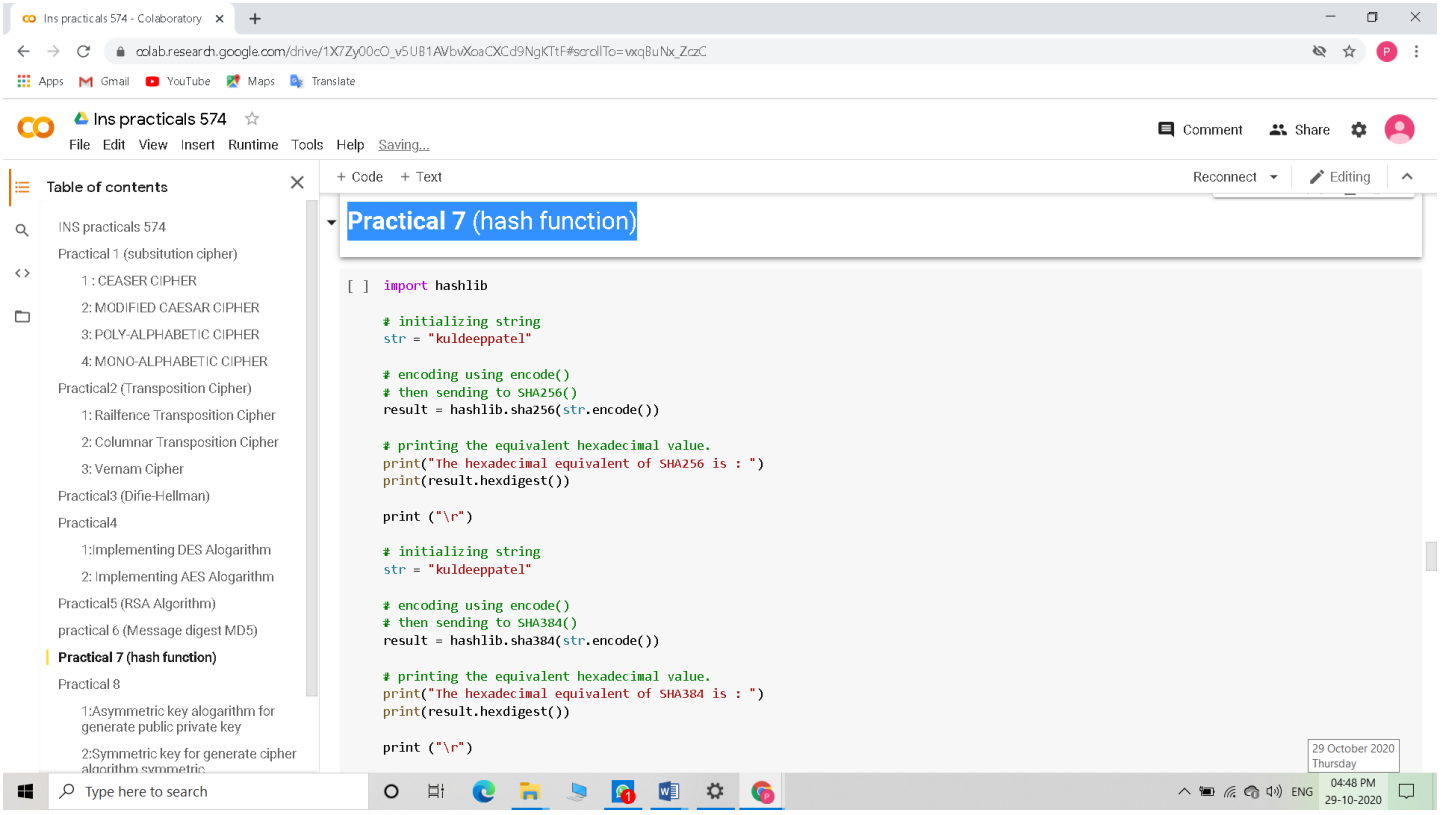
- A **hash function** is a mathematical **function** that converts a numerical input value into another compressed numerical value.
- The input to the **hash function** is of arbitrary length but output is always of fixed length. Values returned by a **hash function** are called message digest or simply **hash** values.

Fixed Length Output (Hash Value)

- Hash function coverts data of arbitrary length to a fixed length. This process is often referred to as **hashing the data**.
- In general, the hash is much smaller than the input data, hence hash functions are sometimes called **compression functions**.
- Since a hash is a smaller representation of a larger data, it is also referred to as a **digest**.
- Hash function with n bit output is referred to as an **n-bit hash function**. Popular hash functions generate values between 160 and 512 bits.



Code :



Ins practicals 574 - Colaboratory

colab.research.google.com/drive/1X7Zy00cO_v5UB1AVbvXoaCXCd9NgKTtF#scrollTo=vxqBuNx_ZczC

Ins practicals 574

File Edit View Insert Runtime Tools Help Saving...

Comment Share

Table of contents

- INS practicals 574
- Practical 1 (subsitution cipher)
  - 1 : CEASER CIPHER
  - 2: MODIFIED CAESAR CIPHER
  - 3: POLY-ALPHABETIC CIPHER
  - 4: MONO-ALPHABETIC CIPHER
- Practical2 (Transposition Cipher)
  - 1: Railfence Transposition Cipher
  - 2: Columnar Transposition Cipher
  - 3: Vernam Cipher
- Practical3 (Difie-Hellman)
- Practical4
  - 1:Implementing DES Alogarithm
  - 2: Implementing AES Alogarithm
- Practical5 (RSA Algorithm)
- practical 6 (Message digest MD5)
- **Practical 7 (hash function)**
- Practical 8
  - 1:Asymmetric key alogarithm for generate public private key
  - 2:Symmetric key for generate cipher algorithm symmetric

+ Code + Text

Reconnect Editing

Practical 7 (hash function)

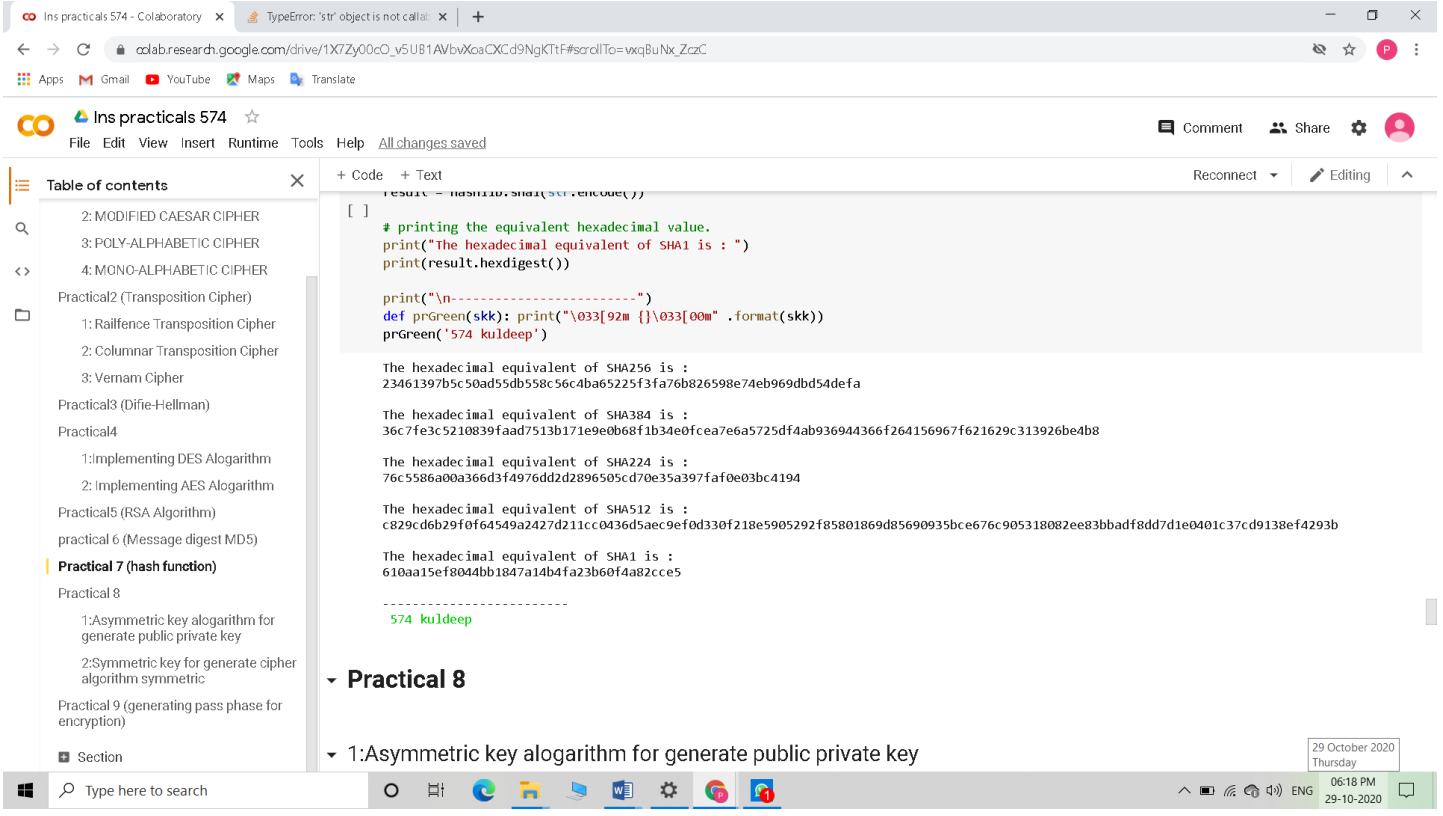
```
[ ]  import hashlib

     # initializing string
     str = "kuldeeppatel"

     # encoding using encode()
     # then sending to SHA256()
     result = hashlib.sha256(str.encode())

     # printing the equivalent hexadecimal value.
     print("The hexadecimal equivalent of SHA256 is : ")
     print(result.hexdigest())

     print ("\r")

     # initializing string
     str = "kuldeeppatel"

     # encoding using encode()
     # then sending to SHA384()
     result = hashlib.sha384(str.encode())

     # printing the equivalent hexadecimal value.
     print("The hexadecimal equivalent of SHA384 is : ")
     print(result.hexdigest())

     print ("\r")
```

29 October 2020 Thursday

Type here to search

ENG 04:48 PM 29-10-2020

Output :



Ins practicals 574 - Colaboratory

TypeError: 'str' object is not callat

colab.research.google.com/drive/1X7Zy00cO_v5UB1AVbvXoaCXCd9NgKTtF#scrollTo=vxqBuNx_ZczC

Ins practicals 574

File Edit View Insert Runtime Tools Help All changes saved

Comment Share

Table of contents

- 2: MODIFIED CAESAR CIPHER
- 3: POLY-ALPHABETIC CIPHER
- 4: MONO-ALPHABETIC CIPHER
- Practical2 (Transposition Cipher)
  - 1: Railfence Transposition Cipher
  - 2: Columnar Transposition Cipher
  - 3: Vernam Cipher
- Practical3 (Difie-Hellman)
- Practical4
  - 1:Implementing DES Alogarithm
  - 2: Implementing AES Alogarithm
- Practical5 (RSA Algorithm)
- practical 6 (Message digest MD5)
- **Practical 7 (hash function)**
- Practical 8
  - 1:Asymmetric key alogarithm for generate public private key
  - 2:Symmetric key for generate cipher algorithm symmetric
- Practical 9 (generating pass phase for encryption)
- Section

+ Code + Text

Reconnect Editing

```
[ ]  # printing the equivalent hexadecimal value.
     print("The hexadecimal equivalent of SHA1 is : ")
     print(result.hexdigest())

     print("\n-------------------------")
     def prGreen(skk): print("\033[92m {}\033[00m" .format(skk))
     prGreen('574 kuldeep')
```

```
The hexadecimal equivalent of SHA256 is :
23461397b5c50ad55db558c56c4ba65225f3fa76b826598e74eb969dbd54defa

The hexadecimal equivalent of SHA384 is :
36c7fe3c5210839faad7513b171e9e0b68f1b34e0fcea7e6a5725df4ab936944366f264156967f621629c313926be4b8

The hexadecimal equivalent of SHA224 is :
76c5586a00a366d3f4976dd2d2896505cd70e35a397faf0e03bc4194

The hexadecimal equivalent of SHA512 is :
c829cd6b29f0f64549a2427d211cc0436d5aec9ef0d330f218e5905292f85801869d85690935bce676c905318082ee83bbadf8dd7d1e0401c37cd9138ef4293b

The hexadecimal equivalent of SHA1 is :
610aa15ef8044bb1847a14b4fa23b60f4a82cce5

-------------------------
 574 kuldeep
```

Practical 8

1:Asymmetric key alogarithm for generate public private key

29 October 2020 Thursday

Type here to search

ENG 06:18 PM 29-10-2020

Colab link :

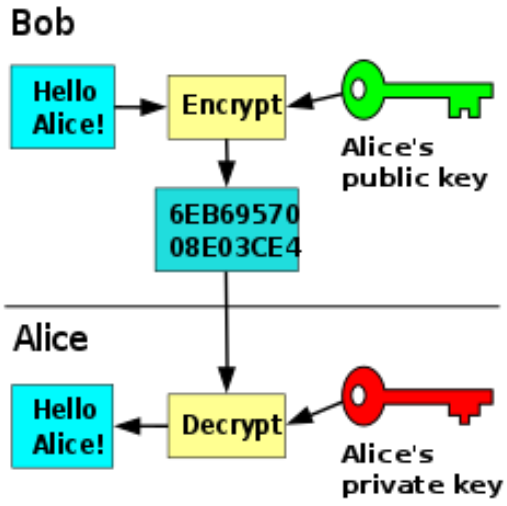
- https://colab.research.google.com/drive/1X7Zy00cO_v5UB1AVbvXoaCXCd9NgKTtF#scrollTo=vxqBuNx_ZczC

Practical 8

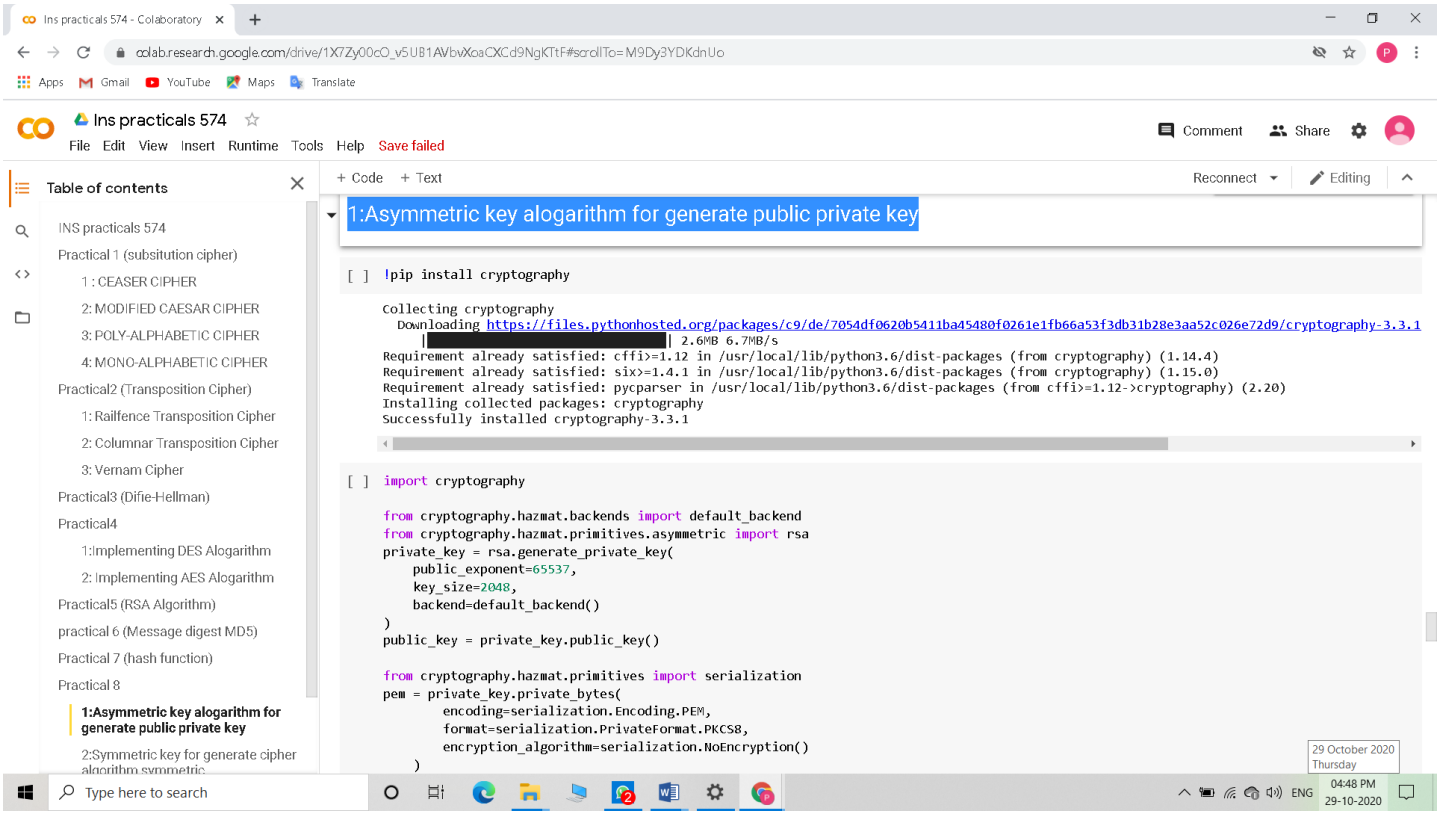
Aim : ASYMMETRIC KEY ALGORITHM FOR PUBLIC PRIVATE KEY CRYPTOGRAPHY

Theory :

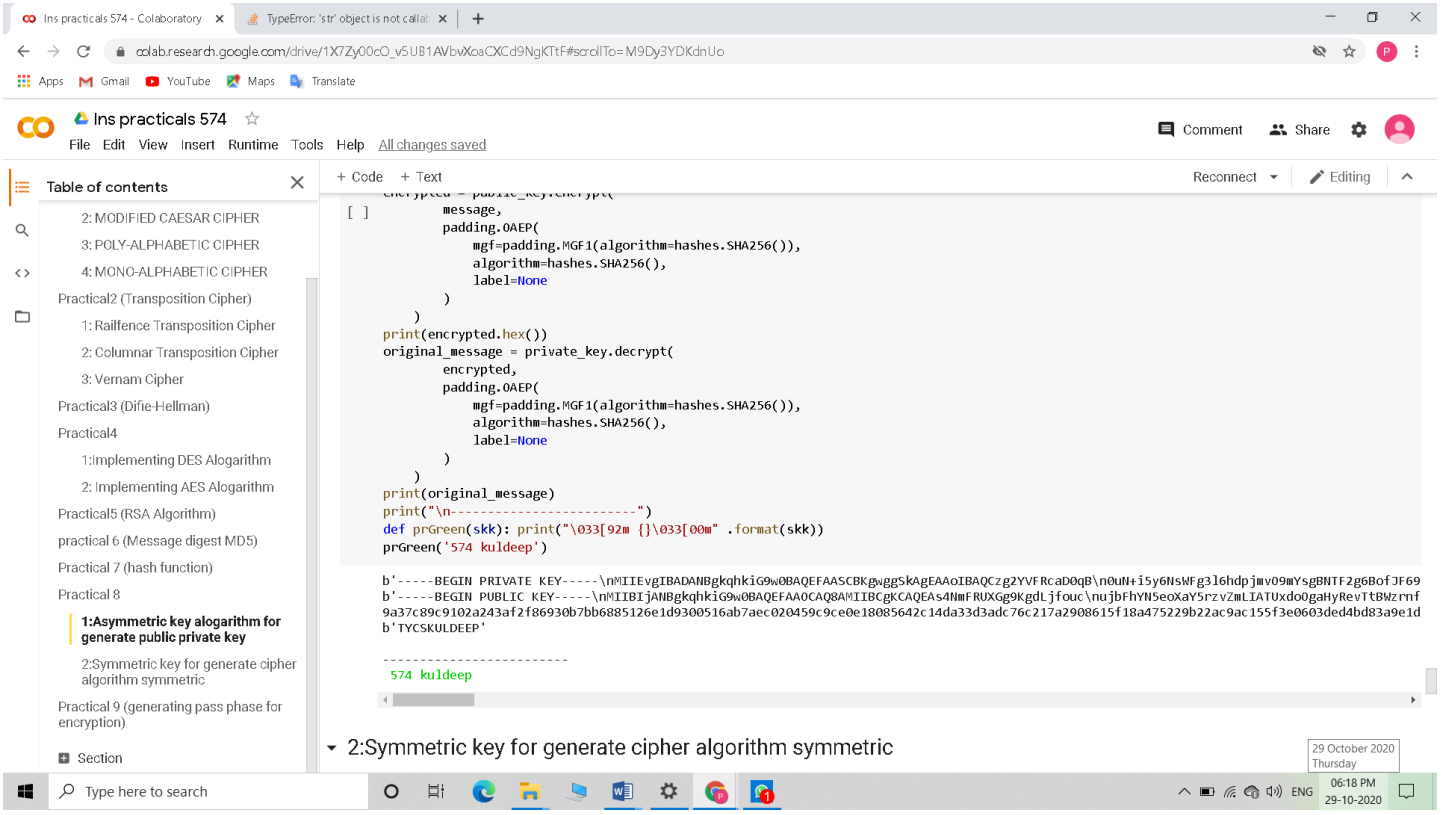
- An unpredictable (typically large and random) number is used to begin generation of an acceptable pair of **keys** suitable for use by an **asymmetric key algorithm**.
- In an **asymmetric key encryption** scheme, anyone can encrypt messages using the **public key**, but only the holder of the paired **private key** can decrypt.



Code :



Output :



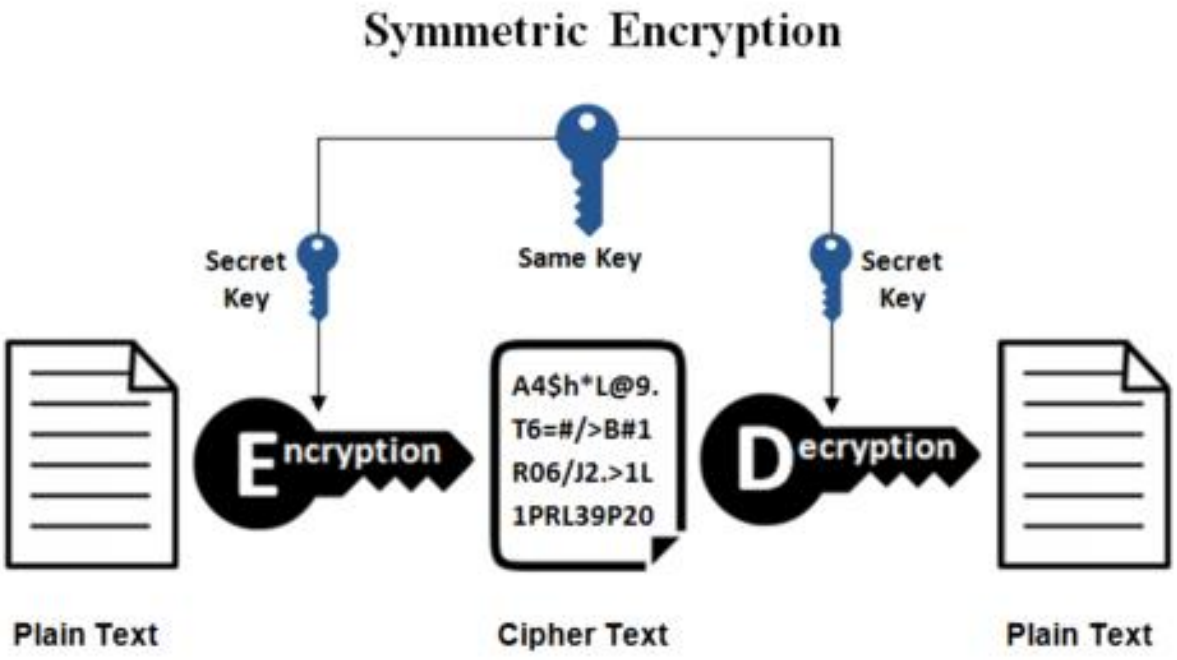
Colab link :

- https://colab.research.google.com/drive/1X7Zy00cO_v5UB1AVbvXoaCXCd9NgKTtF#scrollTo=M9Dy3YDKdnUo

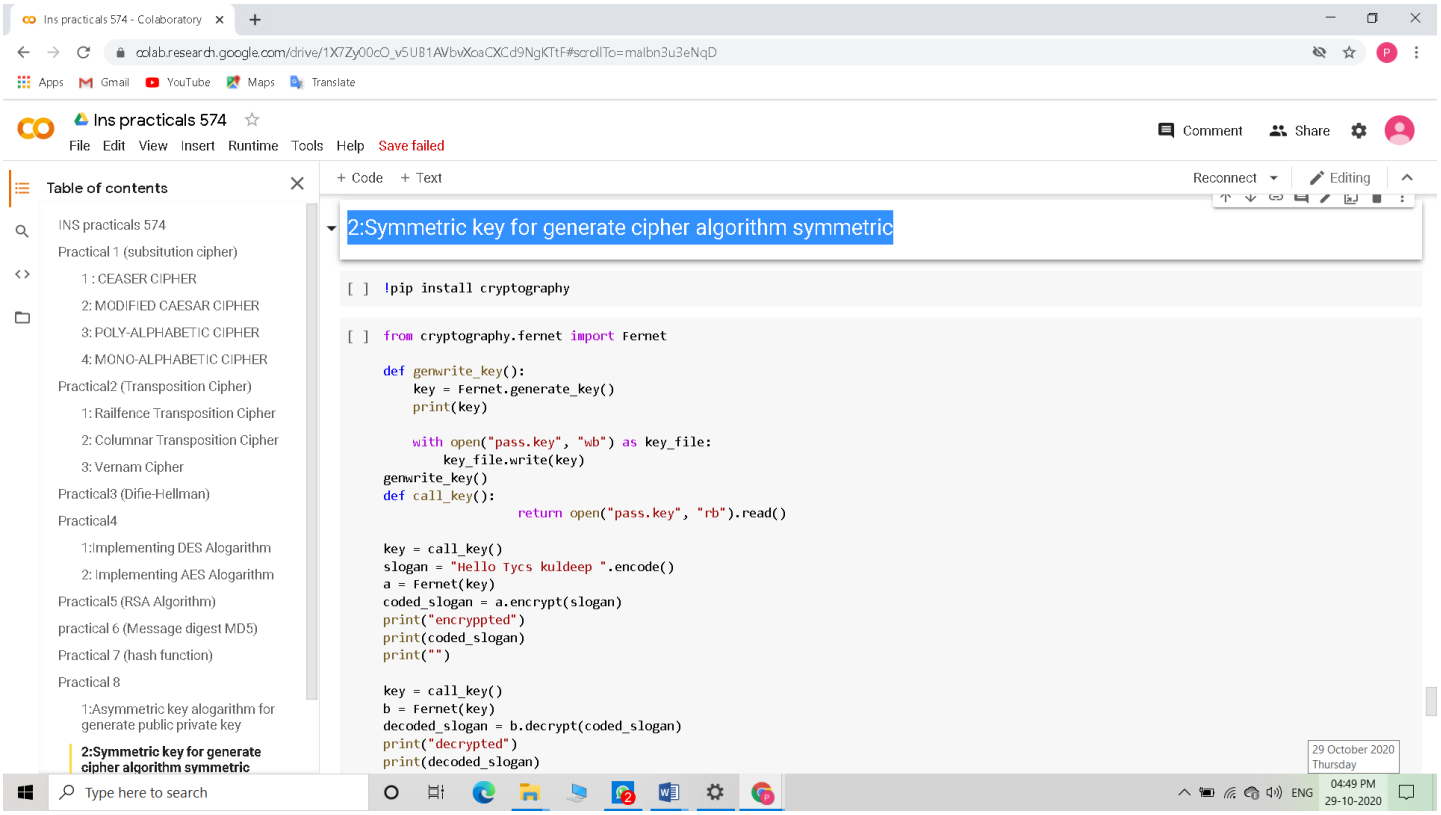
2] SYMMETRIC KEY ALGORITHM

Theory :

- The symmetry of the algorithm comes from the fact that both parties involved share the same key for both encryption and decryption.
- It works similar to a physical door where everyone uses a copy of the same key to both lock and unlock the door.
- A symmetric-key algorithm, just like real doors, requires the distribution and security of *shared* keys.
- Symmetric-key algorithms work by taking the *plaintext* message (i.e., the naturally readable information)
- combining it with a shared key that is input to the algorithm, which outputs the *ciphertext* (i.e., the encrypted text).
- The process works in reverse to decrypt the message. The combined ciphertext and the shared key are input to the algorithm, which outputs the plaintext.

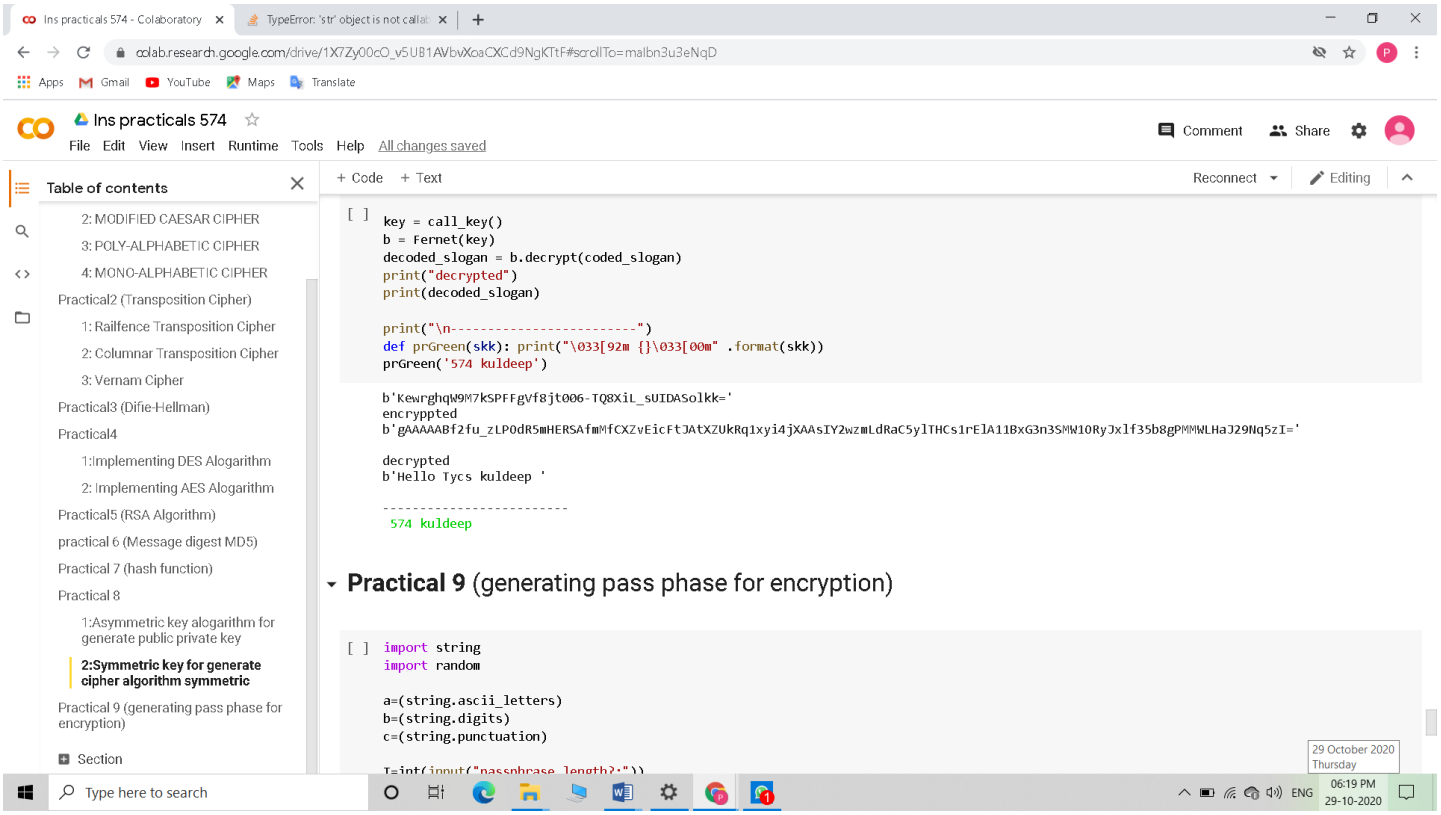


Code :



```
2:Symmetric key for generate cipher algorithm symmetric

!pip install cryptography

from cryptography.fernet import Fernet

def genwrite_key():
    key = Fernet.generate_key()
    print(key)

    with open("pass.key", "wb") as key_file:
        key_file.write(key)
genwrite_key()
def call_key():
                return open("pass.key", "rb").read()

key = call_key()
slogan = "Hello Tycs kuldeep ".encode()
a = Fernet(key)
coded_slogan = a.encrypt(slogan)
print("encryppted")
print(coded_slogan)
print("")

key = call_key()
b = Fernet(key)
decoded_slogan = b.decrypt(coded_slogan)
print("decrypted")
print(decoded_slogan)
```

Output :



```
key = call_key()
b = Fernet(key)
decoded_slogan = b.decrypt(coded_slogan)
print("decrypted")
print(decoded_slogan)

print("\n-------------------------")
def prGreen(skk): print("\033[92m {}\033[00m" .format(skk))
prGreen('574 kuldeep')

b'KewrghqW9M7kSPFFgVf8jt006-TQ8XiL_sUIDASolkk='
encryppted
b'gAAAAABf2fu_zLPOdR5mHERSAfmMfCXZvEicFtJAtXZUkRq1xyi4jXAAsIY2wzmLdRaC5ylTHCs1rElA11BxG3n3SMW1ORyJxlf35b8gPMMWLHaJ29Nq5zI='

decrypted
b'Hello Tycs kuldeep '

-------------------------
 574 kuldeep
```

Practical 9 (generating pass phase for encryption)

```
import string
import random

a=(string.ascii_letters)
b=(string.digits)
c=(string.punctuation)
```

Colab link :

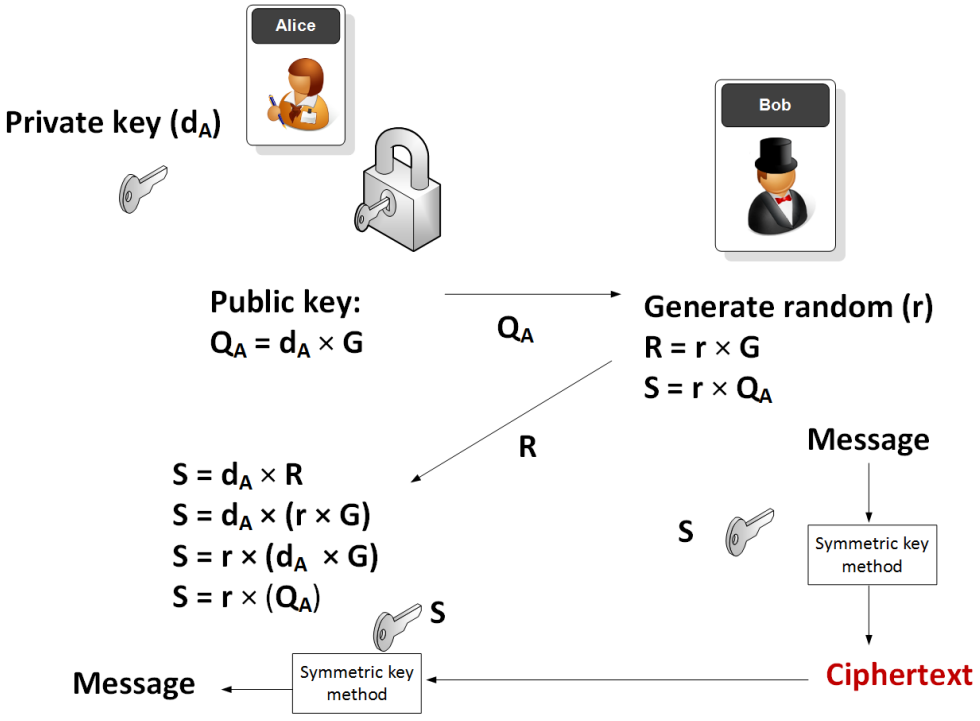
- https://colab.research.google.com/drive/1X7Zy00cO_v5UB1AVbvXoaCXCd9NgKTtF#scrollTo=malbn3u3eNqD

Practical 9

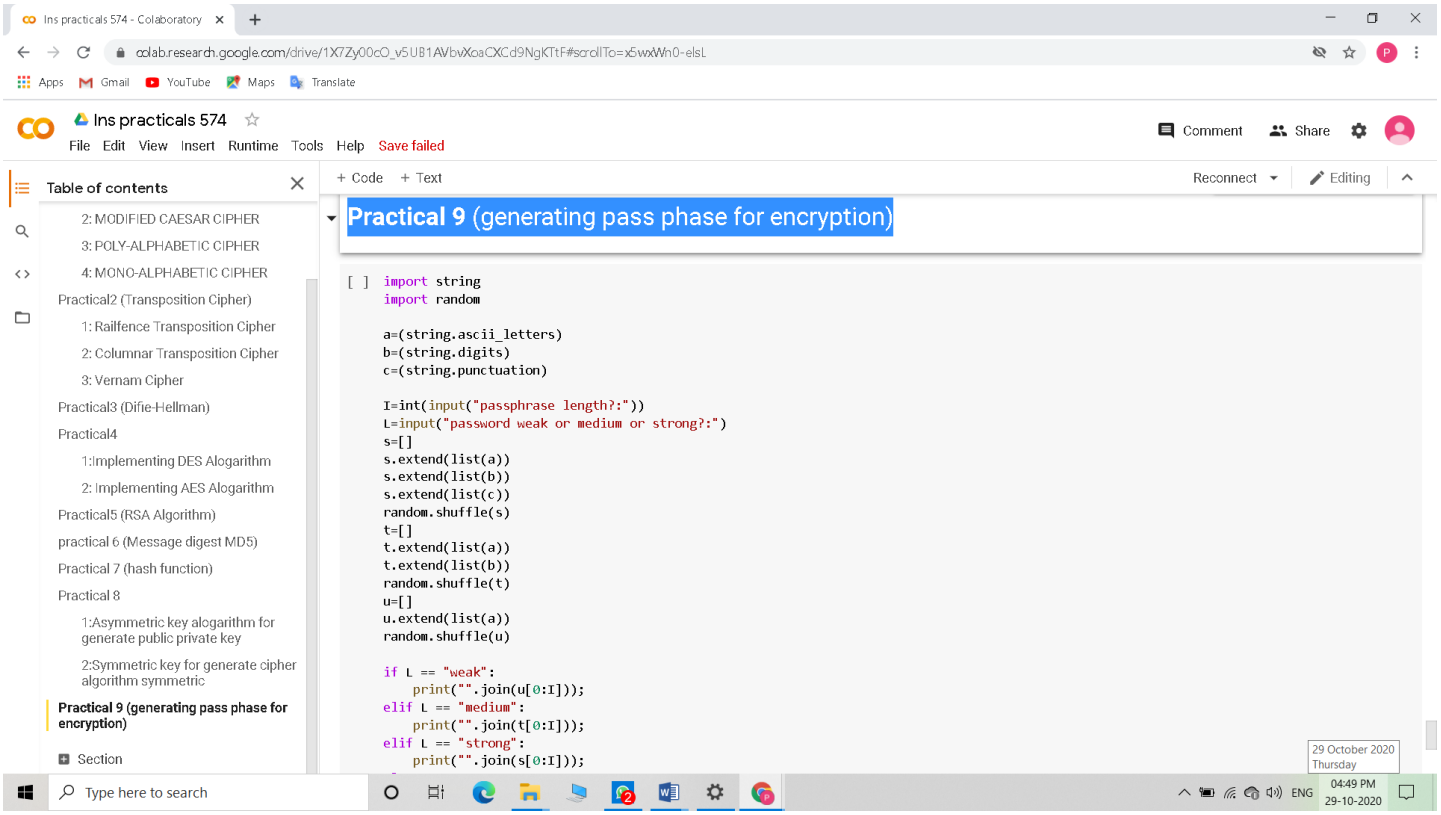
Aim : generating the pass phase for encryption

Theory :

- Passphrase encryption is a “quick-and-dirty” method for encrypting data. Instead of having to manage a private key in a file.
- a passphrase is used to generate a key.
- A passphrase is something a person can remember and type, which eliminates the need to store a key in a file somewhere.
- A passphrase is just like a password, except it’s usually longer. The key is constructed by calculating a message digest of the passphrase.
- The digest value is used to construct a key for a symmetric cipher.

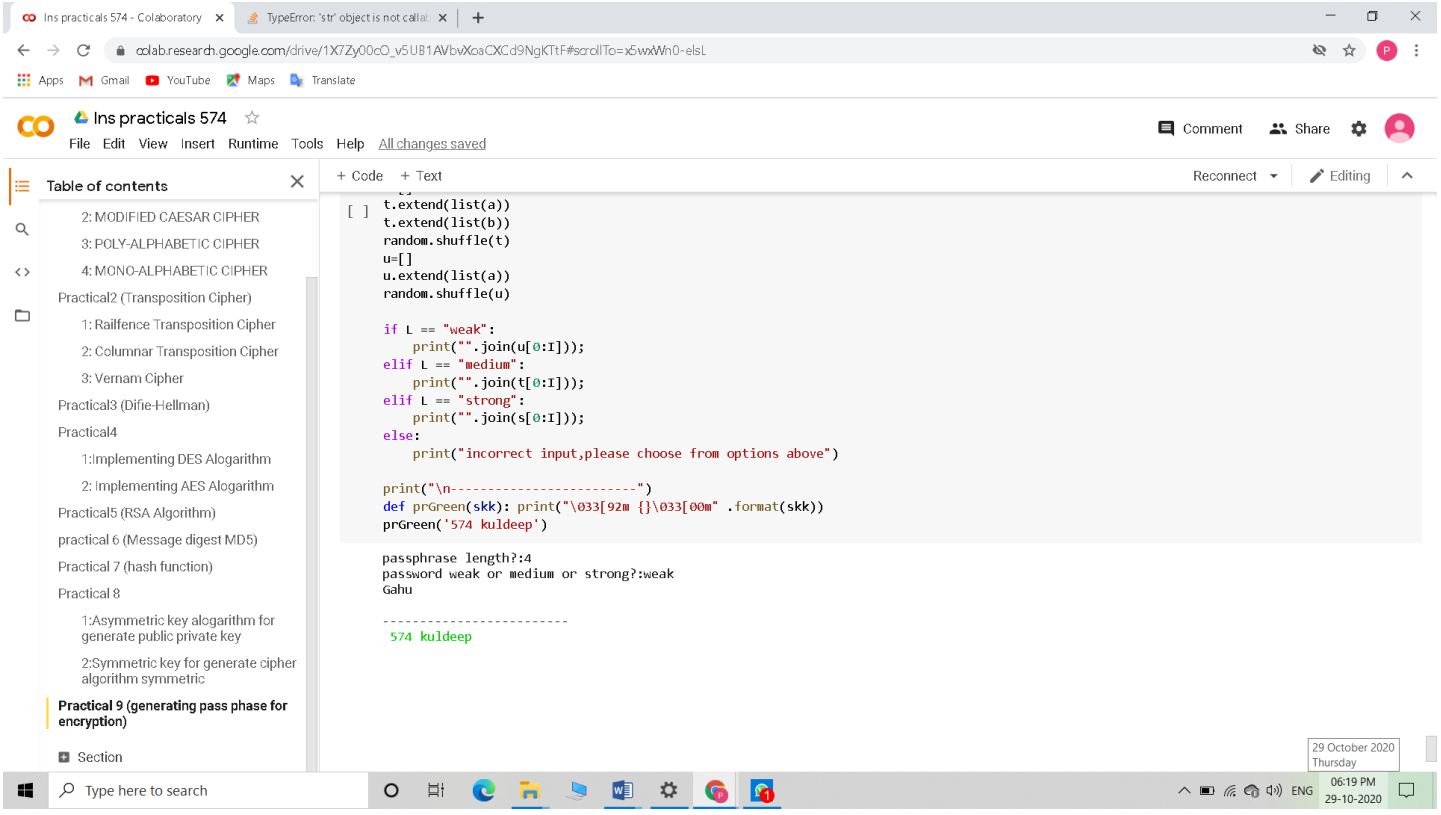


Code :



```
Practical 9 (generating pass phase for encryption)

[ ] import string
    import random

    a=(string.ascii_letters)
    b=(string.digits)
    c=(string.punctuation)

    I=int(input("passphrase length?:"))
    L=input("password weak or medium or strong?:")
    s=[]
    s.extend(list(a))
    s.extend(list(b))
    s.extend(list(c))
    random.shuffle(s)
    t=[]
    t.extend(list(a))
    t.extend(list(b))
    random.shuffle(t)
    u=[]
    u.extend(list(a))
    random.shuffle(u)

    if L == "weak":
        print("".join(u[0:I]));
    elif L == "medium":
        print("".join(t[0:I]));
    elif L == "strong":
        print("".join(s[0:I]));
```

Output :



```
[ ]     t.extend(list(a))
        t.extend(list(b))
        random.shuffle(t)
        u=[]
        u.extend(list(a))
        random.shuffle(u)

        if L == "weak":
            print("".join(u[0:I]));
        elif L == "medium":
            print("".join(t[0:I]));
        elif L == "strong":
            print("".join(s[0:I]));
        else:
            print("incorrect input,please choose from options above")

        print("\n-------------------------")
        def prGreen(skk): print("\033[92m {}\033[00m" .format(skk))
        prGreen('574 kuldeep')

    passphrase length?:4
    password weak or medium or strong?:weak
    Gahu

    -------------------------
     574 kuldeep
```

Colab link :

- https://colab.research.google.com/drive/1X7Zy00cO_v5UB1AVbvXoaCXCd9NgKTtF#scrollTo=x5wxWn0-elsL