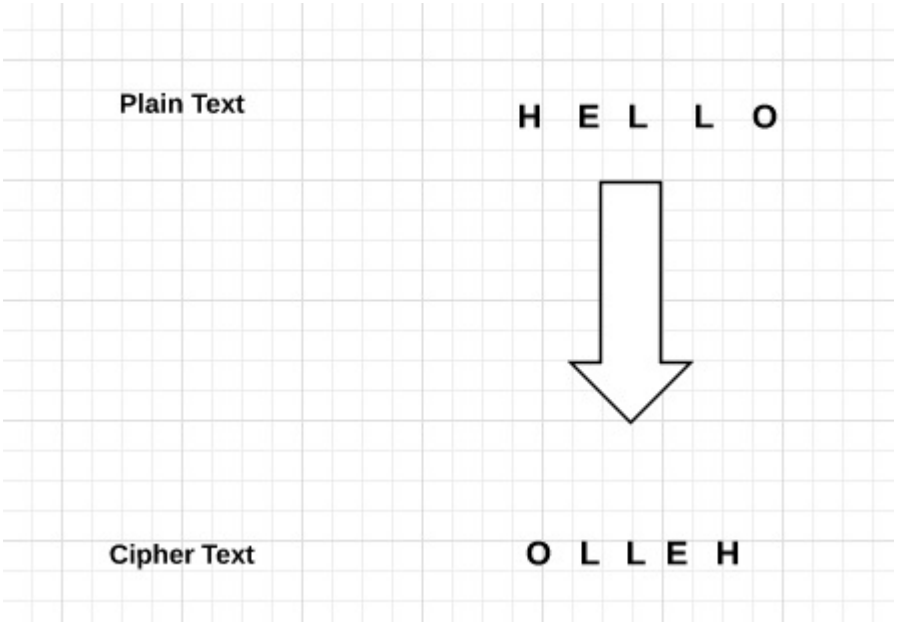
Bài 0: KIẾN THỨC CHUẨN BỊ

https://docs.python.org/3/tutorial/index.html
https://www.pythontutorial.net/python-basics/python-iterables/
https://www.onlinegdb.com/online_python_compiler

BÀI 1: CÁC HỆ MÃ CỔ ĐIỂN

1. Mã đảo ngược - Reverse Cipher:



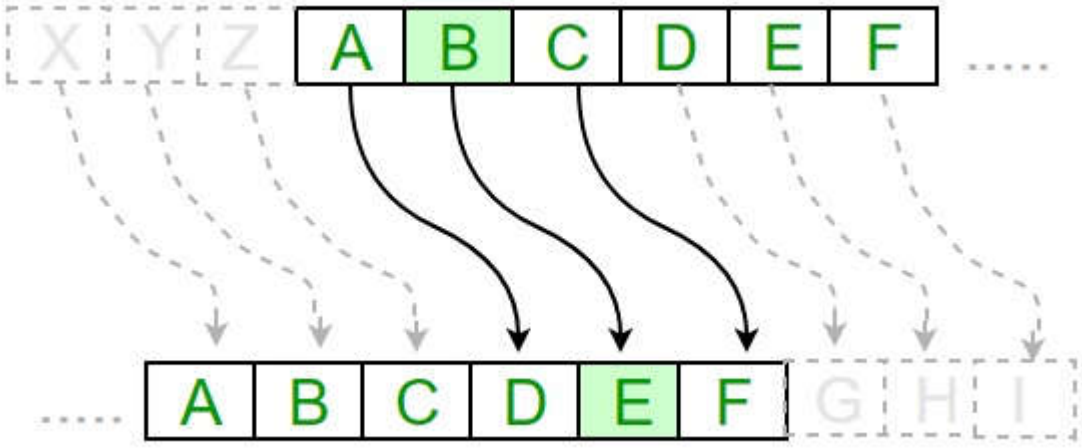
Thuật toán mã đảo ngược:

- Mã hóa: Đảo ngược từng ký tự trong bản rõ thu được bản mã
- Giải mã: Đảo ngược từng ký tự trong bản mã thu được bản rõ

Code:

```
def encrypt (message):
    i = len(message) - 1
    translated = ''
    while i >= 0:
       translated = translated + message[i]
       i = i - 1
    return translated

def decrypt (translated):
    i = len(translated) - 1
    decrypted = ''
    while i >= 0:
       decrypted = decrypted + translated[i]
       i = i - 1
    return decrypted

message = 'This is program to explain reverse cipher.'
translated = encrypt (message)
print("The cipher text is : ", translated)
decrypted = decrypt (translated)
print("The plain text is : ", decrypted)
```

2. Mã Caesar - Caesar Cipher:



Thuật toán mã Caesar:

- Gọi C và P lần lượt là không gian bản mã và bản rõ, với hệ mã Caesar ta có $C \equiv P$; N là số phần tử của bảng chữ cái, ta có $N = 26$. Đánh số các chữ cái từ 0 đến $N - 1$
- Không gian khóa $\boldsymbol{K} = \mathbb{Z}_N$, với mỗi khóa $k \in \boldsymbol{K}$ hàm mã hóa và giải mã tại ký tự có số thứ tự i như sau:
- Mã hóa: $E_k(i) = (i + k) \bmod N$ (ký tự thứ i trở thành ký tự thứ $(i + k) \bmod N$)
- Giải mã: $D_k(i) = (i - k) \bmod N$ (ký tự thứ i trở thành ký tự thứ $(i - k) \bmod N$)

Code:

```
def encrypt(text,k):
    text = text.replace(" ","")
    result = ""
    for i in range(len(text)):
      char = text[i]
      if (char.isupper()):
         result += chr((ord(char) + k - 65) % 26 + 65)
      else:
         result += chr((ord(char) + k - 97) % 26 + 97)
    return result
def decrypt(text,k):
    text = text.replace(" ","")
    result = ""
    for i in range(len(text)):
      char = text[i]
      if (char.isupper()):
         result += chr((ord(char) - k - 65) % 26 + 65)
      else:
         result += chr((ord(char) + k - 97) % 26 + 97)
    return result
text = "CEASER CIPHER DEMO"
k = 4
print ("Key = : " ,k)
c = encrypt(text,k)
print ("Cipher text: ", c )
print ("Plain text: ", decrypt(c,k))
```

3. Mã đổi chỗ

Cho plain text = "**hello world**", phân bố các ký tự trong plain text vào ma trận 2x5:

h	e	l	l	o
w	o	r	l	d

Đọc plain text theo cột ta được cipher text = "**hweolrllod**". Việc giải mã thực hiện hoàn toàn tương tự.

Code:

```
def split_len(seq, length):
   return [seq[i:i + length] for i in range(0, len(seq), length)]

def encrypt(plaintext, key):
    plaintext = plaintext.replace(" ","")
    order = {
      int(val): num for num, val in enumerate(key)
    }
    ciphertext = ''
    for index in sorted(order.keys()):
       for part in split_len(plaintext, len(key)):
          try:ciphertext += part[order[index]]
          except IndexError:
                continue
    return ciphertext

def decrypt(ciphertext, key):
    ciphertext = ciphertext.replace(" ","")
    order = {
      int(val): num for num, val in enumerate(key)
    }
    plaintext = ''
    n = int(len(ciphertext)/len(key))
    for index in sorted(order.keys()):
       for part in split_len(ciphertext, n):
            try:plaintext += part[order[index]]
            except IndexError:
                continue
    return plaintext
k = "12345"
c = encrypt('HELLOWORLDLOVES',k)
print(c)
print(decrypt(c,k))
```

4. Mã thay thế đơn

Quá trình mã hóa của hệ mã thay thế đơn là một phép thay thế mỗi ký tự trong bản rõ thành duy nhất một ký tự trong bản mã với phép thay thế được xem là khóa

VD: Cho khóa

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
p	h	q	g	i	u	m	e	a	y	l	n	o	f	d	x	j	k	r	c	v	s	t	z	w	b

- **Plaintext:** defend the east wall of the castle
- **Ciphertext:** giuifg cei iprc tpnn du cei qprcni

Code:

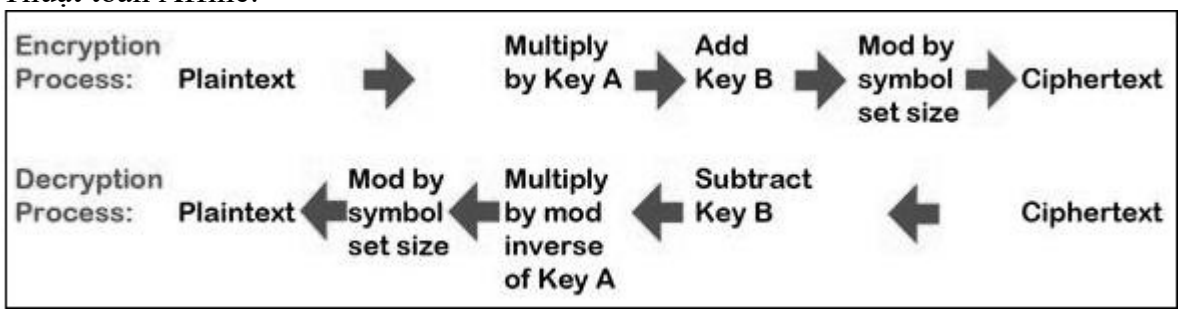
```
import random, sys

def encrypt(message, key):
    translated = ''
    charsA = LETTERS
    charsB = key
    for symbol in message:
        if symbol.upper() in charsA:
            symIndex = charsA.find(symbol.upper())
            if symbol.isupper():
                translated += charsB[symIndex].upper()
            else:
                translated += charsB[symIndex].lower()
        else:
            translated += symbol
    return translated

def decrypt(message, key):
    translated = ''
    charsB = LETTERS
    charsA = key
    for symbol in message:
        if symbol.upper() in charsA:
            symIndex = charsA.find(symbol.upper())
            if symbol.isupper():
                translated += charsB[symIndex].upper()
            else:
                translated += charsB[symIndex].lower()
        else:
            translated += symbol
    return translated

def getRandomKey():
    randomList = list(LETTERS)
    random.shuffle(randomList)
    return ''.join(randomList)

LETTERS = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
message = 'defend the east wall of the castle'
key = ''
key = input("Enter 26 ALPHA key (blank for random key): ")
if key == '':
    key = getRandomKey()

translated = encrypt(message, key)
print('Using key: %s' % (key))
print('Cipher: ' + translated)
print('Plain: ' + decrypt(translated,key))
```

Code: (phiên bản khác)

```
import random, sys

LETTERS = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
def main():
    message = ''
    if len(sys.argv) > 1:
        with open(sys.argv[1], 'r') as f:
            message = f.read()
    else:
        message = input("Enter your message: ")
    mode = input("E for Encrypt, D for Decrypt: ")
    key = ''

    while checkKey(key) is False:
        key = input("Enter 26 ALPHA key (enter for random key): ")
        if key == '':
            key = getRandomKey()
    if checkKey(key) is False:
           print("There is an error in the key or symbol set.")
    translated = translateMessage(message, key, mode)
    print('Using key: %s' % (key))

    if len(sys.argv) > 1:
        fileOut = 'enc.' + sys.argv[1]
        with open(fileOut, 'w') as f:
            f.write(translated)
        print('Success! File written to: %s' % (fileOut))
    else: print('Result: ' + translated)

def checkKey(key):
    keyString = ''.join(sorted(list(key)))
    return keyString == LETTERS
def translateMessage(message, key, mode):
    translated = ''
    charsA = LETTERS
    charsB = key
    if mode == 'D':
        charsA, charsB = charsB, charsA
    for symbol in message:
        if symbol.upper() in charsA:
            symIndex = charsA.find(symbol.upper())
            if symbol.isupper():
                translated += charsB[symIndex].upper()
            else:
                translated += charsB[symIndex].lower()
        else:
            translated += symbol
    return translated
def getRandomKey():
    randomList = list(LETTERS)
    random.shuffle(randomList)
    return ''.join(randomList)
if __name__ == '__main__':
    main()
```

5. Mã Affine

Thuật toán Affine:



Code:

```
def mod_inverse(x,m):
    for n in range(m):
        if (x * n) % m == 1:
            return n
            break
        elif n == m - 1:
            return "Null"
        else:
            continue

class Affine(object):
    DIE = 26
    KEY = (7, 3, mod_inverse(7,26))
    def __init__(self):
        pass
    def encryptChar(self, char):
        K1, K2, kI = self.KEY
        return chr((K1 * (ord(char)-65) + K2) % self.DIE + 65)

    def encrypt(self, string):
        return "".join(map(self.encryptChar, string))

    def decryptChar(self, char):
        K1, K2, KI = self.KEY
        return chr(KI * ((ord(char)-65) - K2) % self.DIE + 65)

    def decrypt(self, string):
        return "".join(map(self.decryptChar, string))

affine = Affine()
p = 'ONAUGUST'
c = affine.encrypt(p)
print (affine.KEY)
print (c)
print(affine.decrypt(c))
```

6. Mã Vigenere https://gist.github.com/dssstr/aedbb5e9f2185f366c6d6b50fad3e4a4

Thuật toánVigenere chia bản rõ P ra thành từng đoạn có độ dài bằng với độ dài khóa, quá trình mã hóa và giải mã được thực hiện như sau:

- Mã hóa: $E_k(P_i) = (P_i + K_i) \bmod 26$
- Giải mã: $D_k(C_i) = (C_i - K_i) \bmod 26$

Mã Vigenere sử dụng nhiều phép thay thế, khóa của hệ mã là một chuỗi M ký tự thay vì là một ký tự. Số thứ tự của các ký tự được đánh số từ 0 đến 25 tương ứng từ A đến Z.
VD: Xét bảng chữ cái tiếng Anh với $N = 26$, giả sử $\boldsymbol{M = 6}$, $\mathcal{k} = "CIPHER"$, hãy mã hóa bản rõ P="THIS CRYPTOSYSTEM IS NOT SECURE"

- $\mathcal{k} = 2, 8, 15, 7, 4, 17$
- P = "THIS CR | YPTOSY | STEM IS | NOT SEC | URE" =
  19 7 8 18 2 17 | 24 15 19 14 18 24 | 18 19 4 12 8 18 | 13 14 19 18 4 2 | 20 17 4

Quá trình mã hóa:

$P = 19\ 07\ 08\ 18\ 02\ 17|24\ 15\ 19\ 14\ 18\ 24|18\ 19\ 04\ 12\ 08\ 18|13\ 14\ 19\ 18\ 04\ 02|20\ 17\ 04$

$\mathcal{k} = 02\ 08\ 15\ 07\ 04\ 17|02\ 08\ 15\ 07\ 04\ 17|02\ 08\ 15\ 07\ 04\ 17|02\ 08\ 15\ 07\ 04\ 17|02\ 08\ 15$

$C = 21\ 15\ 23\ 25\ 06\ 08|00\ 23\ 08\ 21\ 22\ 15|20\ 01\ 19\ 19\ 12\ 09|15\ 22\ 08\ 25\ 08\ 19|22\ 25\ 19$

$= VPXZGI\ AXIVWP\ UBTTMJ\ PWIZIT\ WZT$

Plaintext Letter	Subkey		Ciphertext Letter
C (2)	P (15)	→	R (17)
O (14)	I (8)	→	W (22)
M (12)	Z (25)	→	L (11)
M (12)	Z (25)	→	L (11)
O (14)	A (0)	→	O (14)
N (13)	P (15)	→	C (2)
S (18)	I (8)	→	A (0)
E (4)	Z (25)	→	D (3)
N (13)	Z (25)	→	M (12)
S (18)	A (0)	→	S (18)
E (4)	P (15)	→	T (19)
I (8)	I (8)	→	Q (16)
S (18)	Z (25)	→	R (17)
N (13)	Z (25)	→	M (12)
O (14)	A (0)	→	O (14)
T (19)	P (15)	→	I (8)
S (18)	I (8)	→	A (0)
O (14)	Z (25)	→	N (13)
C (2)	Z (25)	→	B (1)
O (14)	A (0)	→	O (14)
M (12)	P (15)	→	B (1)
M (12)	I (8)	→	U (20)
O (14)	Z (25)	→	N (13)
N (13)	Z (25)	→	M (12)

Code:

```
def encrypt(plaintext, key):
    key_length = len(key)
    key_as_int = [ord(i) for i in key]
    plaintext_int = [ord(i) for i in plaintext]
    ciphertext = ''
    for i in range(len(plaintext_int)):
        value = (plaintext_int[i] + key_as_int[i % key_length]) % 26
        ciphertext += chr(value + 65)
    return ciphertext

def decrypt(ciphertext, key):
    key_length = len(key)
    key_as_int = [ord(i) for i in key]
    ciphertext_int = [ord(i) for i in ciphertext]
    plaintext = ''
    for i in range(len(ciphertext_int)):
        value = (ciphertext_int[i] - key_as_int[i % key_length]) % 26
        plaintext += chr(value + 65)
    return plaintext

p = "THISCRYPTOSYSTEMISNOTSECURE"
k = "CIPHER"
c = encrypt(p,k)
print (c)
print (decrypt(c,k))
```

7. Mã Hill https://gist.github.com/EppuHeilimo/0a901056f9e48a451e0c30a55537ad1b

Bản rõ P được chia thành các chuỗi độ dài $\boldsymbol{M}$, chuyển từng ký tự trong chuỗi thành số thứ tự trong bảng chữ cái dưới dạng vector hàng $\boldsymbol{M}$ chiều, với $\mathcal{k}$ là ma trận vuông kích thước $\boldsymbol{M} \times \boldsymbol{M}$ không suy biến, quá trình mã hóa và giải mã được tiến hành như sau:

- Mã hóa: $C = P * \mathcal{k}$
- Giải mã: $\mathrm{P} = \mathrm{C} * \mathcal{k}^{-1}$

```
   import numpy as np

def encrypt(msg):
    msg = msg.replace(" ", "")    #Thay the khoang trang
    K = make_key()                #Tao va Kiem tra Khoa
    len_check = len(msg) % 2 == 0
    if not len_check:
        msg += "0"
    P = create_matrix_of_integers_from_string(msg)    #Tach plaintext
    msg_len = int(len(msg) / 2)
    encrypted_msg = ""
    for i in range(msg_len):                          #Ma hoa P*K
        row_0 = P[0][i] * K[0][0] + P[1][i] * K[0][1]
        integer = int(row_0 % 26 + 65)
        encrypted_msg += chr(integer)
        row_1 = P[0][i] * K[1][0] + P[1][i] * K[1][1]
        integer = int(row_1 % 26 + 65)
        encrypted_msg += chr(integer)
    return encrypted_msg
```

```
def decrypt(encrypted_msg):
```

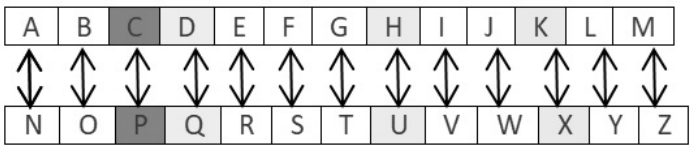
```
    K = make_key()                #Tao va Kiem tra Khoa
    #Tinh dinh thuc
    determinant = K[0][0] * K[1][1] - K[0][1] * K[1][0]
    determinant = determinant % 26
    #Tinh nghich dao cua dinh thuc
    multiplicative_inverse = find_multiplicative_inverse(determinant)
    #Tinh ma tran nghich dao cua Khoa K
    K_inverse = K
    K_inverse[0][0], K_inverse[1][1] = K_inverse[1, 1], K_inverse[0,
0]
    K[0][1] *= -1
    K[1][0] *= -1
    for row in range(2):
        for column in range(2):
            K_inverse[row][column] *= multiplicative_inverse
            K_inverse[row][column] = K_inverse[row][column] % 26
    #Tach cipher text
    C = create_matrix_of_integers_from_string(encrypted_msg)
    msg_len = int(len(encrypted_msg) / 2)
    decrypted_msg = ""
    for i in range(msg_len):      #Giai ma C*K^-1
        column_0 = C[0][i] * K_inverse[0][0] + C[1][i] *
K_inverse[0][1]
        integer = int(column_0 % 26 + 65)
        decrypted_msg += chr(integer)
        column_1 = C[0][i] * K_inverse[1][0] + C[1][i] *
K_inverse[1][1]
        integer = int(column_1 % 26 + 65)
        decrypted_msg += chr(integer)
    if decrypted_msg[-1] == "0":
        decrypted_msg = decrypted_msg[:-1]
    return decrypted_msg

def find_multiplicative_inverse(determinant):  #Tim nghich dao dthuc
    multiplicative_inverse = -1
    for i in range(26):
        inverse = determinant * i
        if inverse % 26 == 1:
            multiplicative_inverse = i
            break
    return multiplicative_inverse

def make_key():      #Tao va kiem tra khoa
    determinant = 0
    K = None
    while True:
        KEY = input("Input 4 letter cipher: ") #Nhap 4 ky tu cach nhau
        KEY = KEY.replace(" ","")
        K = create_matrix_of_integers_from_string(KEY)
        determinant = K[0][0] * K[1][1] - K[0][1] * K[1][0]
        determinant = determinant % 26
        inverse_element = find_multiplicative_inverse(determinant)
        if inverse_element == -1:
            print("Determinant is not relatively prime to 26,
uninvertible key")
```

```
        elif np.amax(K) > 26 and np.amin(K) < 0:
            print("Only a-z characters are accepted")
            print(np.amax(K), np.amin(K))
        else:
            break
    return K

def create_matrix_of_integers_from_string(string): #Tao ma tran Khoa
    integers = [chr_to_int(c) for c in string]
    length = len(integers)
    M = np.zeros((2, int(length / 2)), dtype=np.int32)
    iterator = 0
    for column in range(int(length / 2)):
        for row in range(2):
            M[row][column] = integers[iterator]
            iterator += 1
    return M

def chr_to_int(char):
    char = char.upper()
    integer = ord(char) - 65
    return integer

if __name__ == "__main__":
    msg = input("Message: ")
    encrypted_msg = encrypt(msg)
    print(encrypted_msg)
    decrypted_msg = decrypt(encrypted_msg)
    print(decrypted_msg)
```

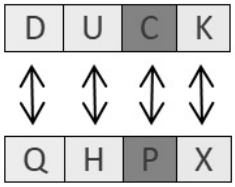
8. Bài tập

Bài 1. Viết chương trình mã hóa và giải mã cho hệ ROT13 với các tính chất sau:

- Bảng ký tự tiếng anh 26 chữ cái;
- Khóa là chuỗi ký tự 26 phần tử, trong đó mỗi phần tử được thay thế bởi phần tử bên phải lớn hơn nó 13 vị trí.



ROT13



Bài 2. Tìm hiểu module base64, sử dụng base64 để viết chương trình mã hóa và giải mã văn bản.

- Mỗi ký tự trong plaintext được đổi thành dạng một byte
- Mỗi chuỗi 6 bit được nhóm thành một nhóm
- Tính giá trị thập phân của chuỗi 6 bit
- Đổi từ số thập phân sang ký tự tương ứng (tra bảng bên dưới).

Value	Char	Value	Char	Value	Char	Value	Char
0	A	16	Q	32	g	48	w
1	B	17	R	33	h	49	x
2	C	18	S	34	i	50	y
3	D	19	T	35	j	51	z
4	E	20	U	36	k	52	0
5	F	21	V	37	l	53	1
6	G	22	W	38	m	54	2
7	H	23	X	39	n	55	3
8	I	24	Y	40	o	56	4
9	J	25	Z	41	p	57	5
10	K	26	a	42	q	58	6
11	L	27	b	43	r	59	7
12	M	28	c	44	s	60	8
13	N	29	d	45	t	61	9
14	O	30	e	46	u	62	+
15	P	31	f	47	v	63	/

VD: Giả sử chúng ta có string là: **rav**

- Binary tương ứng của string trên là : 01110010 01100001 01110110
- Bước đầu tiên là chúng ta chia 3 octet trên thành nhóm 6 bít
- Binary 011100 100110 000101 110110
- Từ đó số thập phân tưng ứng với 4 nhóm mới sẽ là : 28 38 5 54
- Từ bảng trên chúng ta sẽ có được chuỗi ký tự sau khi mã hóa tương ứng như sau:
    - 28 = c
    - 38 = m
    - 5 = F
    - 54 = 2
- Vì thế **rav** sau khi qua base64 sẽ thành **cmF2**

Bài 3. Viết chương trình mã hóa và giải mã của hệ mã XOR (text, key):

Ctrl	Dec	Hex	Char	Code	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
^@	0	00		NUL	32	20		64	40	@	96	60	'
^A	1	01		SOH	33	21	!	65	41	A	97	61	a
^B	2	02		STX	34	22	"	66	42	B	98	62	b
^C	3	03		ETX	35	23	#	67	43	C	99	63	c
^D	4	04		EOT	36	24	$	68	44	D	100	64	d
^E	5	05		ENQ	37	25	%	69	45	E	101	65	e
^F	6	06		ACK	38	26	&	70	46	F	102	66	f
^G	7	07		BEL	39	27	'	71	47	G	103	67	g
^H	8	08		BS	40	28	(	72	48	H	104	68	h
^I	9	09		HT	41	29	)	73	49	I	105	69	i
^J	10	0A		LF	42	2A	*	74	4A	J	106	6A	j
^K	11	0B		VT	43	2B	+	75	4B	K	107	6B	k
^L	12	0C		FF	44	2C	,	76	4C	L	108	6C	l
^M	13	0D		CR	45	2D	-	77	4D	M	109	6D	m
^N	14	0E		SO	46	2E	.	78	4E	N	110	6E	n
^O	15	0F		SI	47	2F	/	79	4F	O	111	6F	o
^P	16	10		DLE	48	30	0	80	50	P	112	70	p
^Q	17	11		DC1	49	31	1	81	51	Q	113	71	q
^R	18	12		DC2	50	32	2	82	52	R	114	72	r
^S	19	13		DC3	51	33	3	83	53	S	115	73	s
^T	20	14		DC4	52	34	4	84	54	T	116	74	t
^U	21	15		NAK	53	35	5	85	55	U	117	75	u
^V	22	16		SYN	54	36	6	86	56	V	118	76	v
^W	23	17		ETB	55	37	7	87	57	W	119	77	w
^X	24	18		CAN	56	38	8	88	58	X	120	78	x
^Y	25	19		EM	57	39	9	89	59	Y	121	79	y
^Z	26	1A		SUB	58	3A	:	90	5A	Z	122	7A	z
^[	27	1B		ESC	59	3B	;	91	5B	[	123	7B	{
^\	28	1C		FS	60	3C	<	92	5C	\	124	7C	|
^]	29	1D		GS	61	3D	=	93	5D	]	125	7D	}
^^	30	1E	▲	RS	62	3E	>	94	5E	^	126	7E	~
^-	31	1F	▼	US	63	3F	?	95	5F	_	127	7F	⌂

* ASCII code 127 has the code DEL. Under MS-DOS, this code has the same effect as ASCII 8 (BS). The DEL code can be generated by the CTRL + BKSP key.

Chú ý:

\a	07	Alert (Beep, Bell) (added in C89)[1]
\b	08	Backspace
\e	1B	Escape character
\f	0C	Formfeed Page Break
\n	0A	Newline (Line Feed); see notes below
\r	0D	Carriage Return
\t	09	Horizontal Tab
\v	0B	Vertical Tab
\\	5C	Backslash
\'	27	Apostrophe or single quotation mark
\"	22	Double quotation mark
\?	3F	Question mark (used to avoid trigraphs)

Với bản rõ P và khóa K, chuyển các ký tự trong bản rõ P và khóa K thành byte mã ASCII, quá trình mã hóa và giải mã được thực hiện:

- Mã hóa: $C = P \oplus K$
- Giải mã: $P = C \oplus K$

VD: Cho bản rõ $P = "GOOD"$, khóa $K = "ABC"$, quá trình mã hóa được thực hiện như sau:
Ta có:

Ký tự	Mã ASCII (hex)	Bytes
A	41	0100 0001
B	42	0100 0010
C	43	0100 0011
G	47	0100 0111
O	4F	0100 1111
D	44	0100 0100

P	0100 0111	0100 1111	0100 1111	0100 0100
K	0100 0001	0100 0010	0100 0011	0100 0001
C	0000 0110	0000 1101	0000 1100	0000 0101
C (hex)	06	0D	0C	05

Bài 4. Viết chương trình mã hóa và giải mã cho hệ mã nhân

Cho bản rõ P là chuỗi ký tự lấy từ bảng chữ cái 26 phần tử (được đánh chỉ số từ 0 đến 25) và khóa K là một số nguyên, trong đó (K,26)=1. Tương tự quá trình mã hóa và giải mã của hệ mã Caesar nhưng ta thay phép cộng modulo thành phép nhân modulo:

- Mã hóa: $C = P * k \ (mod\ 26)$
- Giải mã: $P = C * k^{-1} \ (mod\ 26)$, trong đó k^{-1} là nghịch đảo của k

VD: Cho P = “ONAUGUST”, k = 7
Ta có C = “UNAKQKWD”
Quá trình giải mã thực hiện ngược lại với nghịch đảo khóa k^{-1} được tìm theo thuật toán Euclide mở rộng (như trong mã Affine).

Plaintext Symbol	Number	Encryption with Key 7	Ciphertext Symbol
A	0	(0 * 7) % 26 = 0	A
B	1	(1 * 7) % 26 = 7	H
C	2	(2 * 7) % 26 = 14	O
D	3	(3 * 7) % 26 = 21	V
E	4	(4 * 7) % 26 = 2	C
F	5	(5 * 7) % 26 = 9	J
G	6	(6 * 7) % 26 = 16	Q
H	7	(7 * 7) % 26 = 23	X
I	8	(8 * 7) % 26 = 4	E
J	9	(9 * 7) % 26 = 11	L
K	10	(10 * 7) % 26 = 18	S
L	11	(11 * 7) % 26 = 25	Z
M	12	(12 * 7) % 26 = 6	G
N	13	(13 * 7) % 26 = 13	N
O	14	(14 * 7) % 26 = 20	U
P	15	(15 * 7) % 26 = 1	B
Q	16	(16 * 7) % 26 = 8	I
R	17	(17 * 7) % 26 = 15	P
S	18	(18 * 7) % 26 = 22	W
T	19	(19 * 7) % 26 = 3	D
U	20	(20 * 7) % 26 = 10	K
V	21	(21 * 7) % 26 = 17	R
W	22	(22 * 7) % 26 = 24	Y
X	23	(23 * 7) % 26 = 5	F
Y	24	(24 * 7) % 26 = 12	M
Z	25	(25 * 7) % 26 = 19	T

Bài 5. Sử dụng hàm Fernet trong thư viện cryptography, viết chương trình mã hóa và giải mã một chuỗi ký tự.

Bài 6. Viết chương trình thám mã hệ mã Caesar

Sử dụng module base64, viết chương trình mã hóa và giải mã văn bản

```
import base64

message = "Cipher is fun"
message_bytes = message.encode('ascii')
base64_bytes = base64.b64encode(message_bytes)
base64_message = base64_bytes.decode('ascii')
print(base64_message)

plain_bytes = base64.b64decode(base64_bytes)
plain_text = plain_bytes.decode('ascii')
print(plain_text)
```

Viết chương trình mã hóa và giải mã của hệ mã XOR (text, key)

```
from itertools import cycle
import base64

def xor_encrypt_string(data, key):
    xored = ''.join(chr(ord(x) ^ ord(y)) for (x,y) in zip(data,
cycle(key)))
    xored = xored.encode('ascii')
```

```
    xored = base64.encodestring(xored).strip()
    return xored

def xor_decrypt_string(data, key):
    data = base64.decodestring(data)
    data = data.decode('ascii')
    xored = ''.join(chr(ord(x) ^ ord(y)) for (x,y) in zip(data,
cycle(key)))
    return xored

key = 'cipher'
secret_data = "GoodLife"
c = xor_encrypt_string(secret_data, key)
print("The cipher text is")
print (c)
print("The plain text fetched")
print (xor_decrypt_string(c,key))
```

Viết chương trình mã hóa và giải mã cho hệ mã nhân

```
def mod_inverse(x,m):
    for n in range(m):
        if (x * n) % m == 1:
            return n
            break
        elif n == m - 1:
            return "Null"
        else:
            continue

def encryptChar(char, K1):
    return chr((K1 * (ord(char)-65) ) % 26 + 65)

def encrypt(string, KEY):
    return "".join(encryptChar(c,KEY) for c in string)

def decryptChar(char, KI):
    return chr(KI * ((ord(char)-65) ) % 26 + 65)

def decrypt(string, KEY):
    KI = mod_inverse (KEY,26)
    return "".join(decryptChar(c,KI) for c in string)

p = 'ONAUGUST'
KEY = 7
c = encrypt(p,KEY)
print (c)
print(decrypt(c,KEY))
```

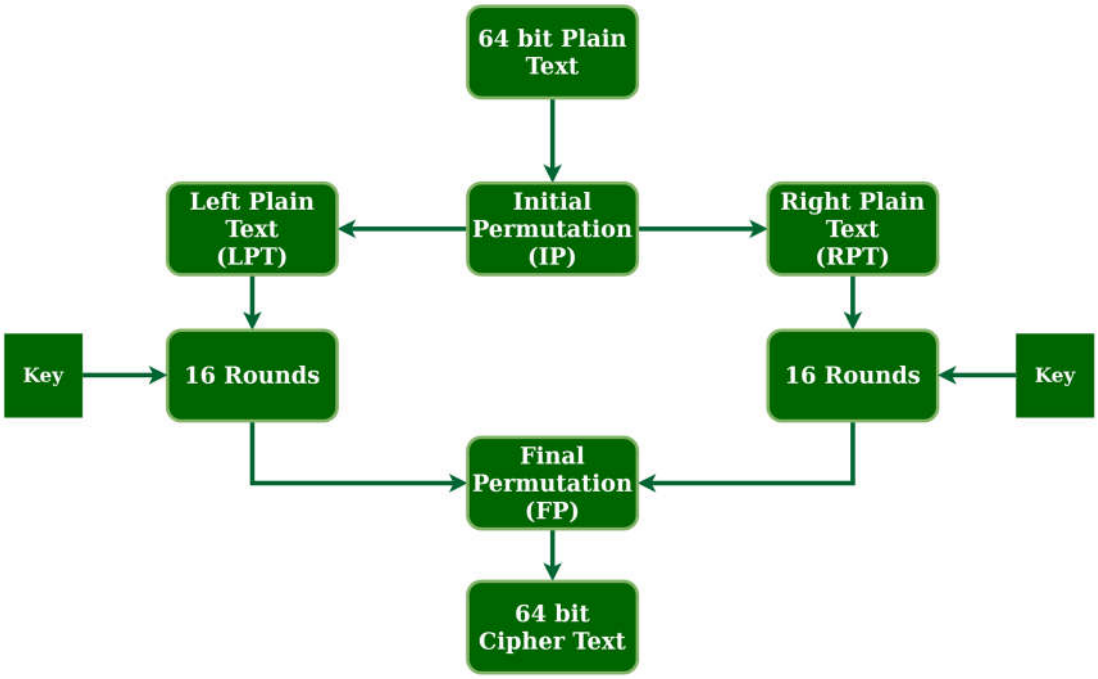
Viết chương trình thám mã hệ mã Caesar

```
message = 'GIEWIVrGMTLIVrHIQS' #encrypted message
LETTERS = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'

for key in range(len(LETTERS)):
   translated = ''
   for symbol in message:
```

```
      if symbol in LETTERS:
         num = LETTERS.find(symbol)
         num = num - key
         if num < 0:
            num = num + len(LETTERS)
         translated = translated + LETTERS[num]
      else:
         translated = translated + symbol
print('Hacking key #%s: %s' % (key, translated))
```

BÀI 2: CÁC HỆ MÃ KHỐI

1. Mã DES




```
# Đổi Thập lục sang Nhị phân
def hex2bin(s):
    mp = {'0' : "0000",
          '1' : "0001",
          '2' : "0010",
          '3' : "0011",
          '4' : "0100",
          '5' : "0101",
          '6' : "0110",
          '7' : "0111",
          '8' : "1000",
          '9' : "1001",
          'A' : "1010",
          'B' : "1011",
          'C' : "1100",
          'D' : "1101",
          'E' : "1110",
          'F' : "1111" }
    bin = ""
    for i in range(len(s)):
        bin = bin + mp[s[i]]
    return bin

# Đổi Nhị phân sang Thập lục
def bin2hex(s):
    mp = {"0000" : '0',
          "0001" : '1',
          "0010" : '2',
          "0011" : '3',
          "0100" : '4',
```

```
          "0101" : '5',
          "0110" : '6',
          "0111" : '7',
          "1000" : '8',
          "1001" : '9',
          "1010" : 'A',
          "1011" : 'B',
          "1100" : 'C',
          "1101" : 'D',
          "1110" : 'E',
          "1111" : 'F' }
    hex = ""
    for i in range(0,len(s),4):
        ch = ""
        ch = ch + s[i]
        ch = ch + s[i + 1]
        ch = ch + s[i + 2]
        ch = ch + s[i + 3]
        hex = hex + mp[ch]

    return hex

# Đổi Nhị phân sang Thập phân
def bin2dec(binary):

    binary1 = binary
    decimal, i, n = 0, 0, 0
    while(binary != 0):
        dec = binary % 10
        decimal = decimal + dec * pow(2, i)
        binary = binary//10
        i += 1
    return decimal

# Đổi Thập phân sang Nhị phân
def dec2bin(num):
    res = bin(num).replace("0b", "")
    if(len(res)%4 != 0):
        div = len(res) / 4
        div = int(div)
        counter =(4 * (div + 1)) - len(res)
        for i in range(0, counter):
            res = '0' + res
    return res

# Hàm hoán vị
def permute(k, arr, n):
    permutation = ""
    for i in range(0, n):
        permutation = permutation + k[arr[i] - 1]
    return permutation

# Hàm dịch vòng trái
def shift_left(k, nth_shifts):
    s = ""
```

```
    for i in range(nth_shifts):
        for j in range(1,len(k)):
            s = s + k[j]
        s = s + k[0]
        k = s
        s = ""
    return k

# Tính xor hai chuỗi nhị phân số a và b
def xor(a, b):
    ans = ""
    for i in range(len(a)):
        if a[i] == b[i]:
            ans = ans + "0"
        else:
            ans = ans + "1"
    return ans

# Bảng hoán vị đầu IP
initial_perm = [58, 50, 42, 34, 26, 18, 10, 2,
                60, 52, 44, 36, 28, 20, 12, 4,
                62, 54, 46, 38, 30, 22, 14, 6,
                64, 56, 48, 40, 32, 24, 16, 8,
                57, 49, 41, 33, 25, 17, 9, 1,
                59, 51, 43, 35, 27, 19, 11, 3,
                61, 53, 45, 37, 29, 21, 13, 5,
                63, 55, 47, 39, 31, 23, 15, 7]

# Hàm mở rộng Expansion
exp_d = [32, 1 , 2 , 3 , 4 , 5 , 4 , 5,
         6 , 7 , 8 , 9 , 8 , 9 , 10, 11,
         12, 13, 12, 13, 14, 15, 16, 17,
         16, 17, 18, 19, 20, 21, 20, 21,
         22, 23, 24, 25, 24, 25, 26, 27,
         28, 29, 28, 29, 30, 31, 32, 1 ]

# Hoán vị P (của hàm Feistel)
per = [ 16,  7, 20, 21,
        29, 12, 28, 17,
         1, 15, 23, 26,
         5, 18, 31, 10,
         2,  8, 24, 14,
        32, 27,  3,  9,
        19, 13, 30,  6,
        22, 11,  4, 25 ]

# Bảng S-box
sbox =  [[[14, 4, 13, 1, 2, 15, 11, 8, 3, 10, 6, 12, 5, 9, 0, 7],
          [ 0, 15, 7, 4, 14, 2, 13, 1, 10, 6, 12, 11, 9, 5, 3, 8],
          [ 4, 1, 14, 8, 13, 6, 2, 11, 15, 12, 9, 7, 3, 10, 5, 0],
          [15, 12, 8, 2, 4, 9, 1, 7, 5, 11, 3, 14, 10, 0, 6, 13 ]],

         [[15, 1, 8, 14, 6, 11, 3, 4, 9, 7, 2, 13, 12, 0, 5, 10],
            [3, 13, 4, 7, 15, 2, 8, 14, 12, 0, 1, 10, 6, 9, 11, 5],
            [0, 14, 7, 11, 10, 4, 13, 1, 5, 8, 12, 6, 9, 3, 2, 15],
```

```
           [13, 8, 10, 1, 3, 15, 4, 2, 11, 6, 7, 12, 0, 5, 14, 9 ]],

         [ [10, 0, 9, 14, 6, 3, 15, 5, 1, 13, 12, 7, 11, 4, 2, 8],
           [13, 7, 0, 9, 3, 4, 6, 10, 2, 8, 5, 14, 12, 11, 15, 1],
           [13, 6, 4, 9, 8, 15, 3, 0, 11, 1, 2, 12, 5, 10, 14, 7],
            [1, 10, 13, 0, 6, 9, 8, 7, 4, 15, 14, 3, 11, 5, 2, 12 ]],

          [ [7, 13, 14, 3, 0, 6, 9, 10, 1, 2, 8, 5, 11, 12, 4, 15],
           [13, 8, 11, 5, 6, 15, 0, 3, 4, 7, 2, 12, 1, 10, 14, 9],
           [10, 6, 9, 0, 12, 11, 7, 13, 15, 1, 3, 14, 5, 2, 8, 4],
            [3, 15, 0, 6, 10, 1, 13, 8, 9, 4, 5, 11, 12, 7, 2, 14] ],

          [ [2, 12, 4, 1, 7, 10, 11, 6, 8, 5, 3, 15, 13, 0, 14, 9],
           [14, 11, 2, 12, 4, 7, 13, 1, 5, 0, 15, 10, 3, 9, 8, 6],
            [4, 2, 1, 11, 10, 13, 7, 8, 15, 9, 12, 5, 6, 3, 0, 14],
           [11, 8, 12, 7, 1, 14, 2, 13, 6, 15, 0, 9, 10, 4, 5, 3 ]],

         [ [12, 1, 10, 15, 9, 2, 6, 8, 0, 13, 3, 4, 14, 7, 5, 11],
           [10, 15, 4, 2, 7, 12, 9, 5, 6, 1, 13, 14, 0, 11, 3, 8],
            [9, 14, 15, 5, 2, 8, 12, 3, 7, 0, 4, 10, 1, 13, 11, 6],
            [4, 3, 2, 12, 9, 5, 15, 10, 11, 14, 1, 7, 6, 0, 8, 13] ],

          [ [4, 11, 2, 14, 15, 0, 8, 13, 3, 12, 9, 7, 5, 10, 6, 1],
           [13, 0, 11, 7, 4, 9, 1, 10, 14, 3, 5, 12, 2, 15, 8, 6],
            [1, 4, 11, 13, 12, 3, 7, 14, 10, 15, 6, 8, 0, 5, 9, 2],
            [6, 11, 13, 8, 1, 4, 10, 7, 9, 5, 0, 15, 14, 2, 3, 12] ],

         [ [13, 2, 8, 4, 6, 15, 11, 1, 10, 9, 3, 14, 5, 0, 12, 7],
            [1, 15, 13, 8, 10, 3, 7, 4, 12, 5, 6, 11, 0, 14, 9, 2],
            [7, 11, 4, 1, 9, 12, 14, 2, 0, 6, 10, 13, 15, 3, 5, 8],
            [2, 1, 14, 7, 4, 10, 8, 13, 15, 12, 9, 0, 3, 5, 6, 11] ] ]

# Bảng hoán vị cuối FP
final_perm = [ 40, 8, 48, 16, 56, 24, 64, 32,
               39, 7, 47, 15, 55, 23, 63, 31,
               38, 6, 46, 14, 54, 22, 62, 30,
               37, 5, 45, 13, 53, 21, 61, 29,
               36, 4, 44, 12, 52, 20, 60, 28,
               35, 3, 43, 11, 51, 19, 59, 27,
               34, 2, 42, 10, 50, 18, 58, 26,
               33, 1, 41, 9, 49, 17, 57, 25 ]

def encrypt(pt, rkb, rk):
    pt = hex2bin(pt)

    # Hoán vị đầu
    pt = permute(pt, initial_perm, 64)
    print("After initial permutation", bin2hex(pt))

    # Phân chia thành nửa trái và nửa phải
    left = pt[0:32]
    right = pt[32:64]
    for i in range(0, 16):
        #  Nửa phải qua hàm mở rộng (32 thành 48)
        right_expanded = permute(right, exp_d, 48)
```

```
        # XOR RoundKey[i] và right_expanded
        xor_x = xor(right_expanded, rkb[i])

        # Qua S-boxex
        sbox_str = ""
        for j in range(0, 8):
            row = bin2dec(int(xor_x[j * 6] + xor_x[j * 6 + 5]))
            col = bin2dec(int(xor_x[j * 6 + 1] + xor_x[j * 6 + 2] +
xor_x[j * 6 + 3] + xor_x[j * 6 + 4]))
            val = sbox[j][row][col]
            sbox_str = sbox_str + dec2bin(val)

        # Hoán vị P
        sbox_str = permute(sbox_str, per, 32)

        # XOR left và sbox_str
        result = xor(left, sbox_str)
        left = result

        # Đổi chỗ (vòng lặp cuối)
        if(i != 15):
            left, right = right, left
        print("Round ", i + 1, " ", bin2hex(left), " ",
bin2hex(right), " ", rk[i])

    # Kết hợp nửa trái và nửa phải lại
    combine = left + right

    # Hoán vị cuối FP
    cipher_text = permute(combine, final_perm, 64)
    return cipher_text

pt = "0123456789ABCDEF"
key = "133457799BBCDFF0"

# Sinh khóa
key = hex2bin(key)

# Bảng PC1
keyp = [57, 49, 41, 33, 25, 17, 9,
        1, 58, 50, 42, 34, 26, 18,
        10, 2, 59, 51, 43, 35, 27,
        19, 11, 3, 60, 52, 44, 36,
        63, 55, 47, 39, 31, 23, 15,
        7, 62, 54, 46, 38, 30, 22,
        14, 6, 61, 53, 45, 37, 29,
        21, 13, 5, 28, 20, 12, 4 ]

# Qua bảng PC1 lấy 56 bit từ 64 bit của khóa
key = permute(key, keyp, 56)

# Số lượng bit dịch vòng
shift_table = [1, 1, 2, 2,
               2, 2, 2, 2,
```

```
                1, 2, 2, 2,
                2, 2, 2, 1 ]

# Bảng PC2: Nén 56 bit thành 48 bit
key_comp = [14, 17, 11, 24, 1, 5,
            3, 28, 15, 6, 21, 10,
            23, 19, 12, 4, 26, 8,
            16, 7, 27, 20, 13, 2,
            41, 52, 31, 37, 47, 55,
            30, 40, 51, 45, 33, 48,
            44, 49, 39, 56, 34, 53,
            46, 42, 50, 36, 29, 32 ]

# Phân chia khóa thành nửa trái nửa phải
left = key[0:28]    # rkb for RoundKeys in binary
right = key[28:56]  # rk for RoundKeys in hexadecimal

rkb = []
rk  = []
for i in range(0, 16):
    # Dịch vòng trái theo số lượng bit của vòng
    left = shift_left(left, shift_table[i])
    right = shift_left(right, shift_table[i])

    # Kết hợp nủa trái và phải
    combine_str = left + right

    # Qua PC2: Nén 56 bit thành 48 bit
    round_key = permute(combine_str, key_comp, 48)

    rkb.append(round_key)
    rk.append(bin2hex(round_key))

print("Encryption")
cipher_text = bin2hex(encrypt(pt, rkb, rk))
print("Cipher Text : ",cipher_text)

print("Decryption")
rkb_rev = rkb[::-1]
rk_rev = rk[::-1]
text = bin2hex(encrypt(cipher_text, rkb_rev, rk_rev))
print("Plain Text : ",text)
```

2. Mã AES

BÀI 3: CÁC HỆ MÃ KHÓA CÔNG KHAI

1. Mã RSA

```
from Crypto.PublicKey import RSA
from Crypto.Cipher import PKCS1_OAEP
import binascii

keyPair = RSA.generate(2048)

pubKey = keyPair.publickey()
print(f"Public key: (n={hex(pubKey.n)}, e={hex(pubKey.e)})")
pubKeyPEM = pubKey.exportKey()
print(pubKeyPEM.decode('ascii'))

print(f"Private key: (n={hex(pubKey.n)}, d={hex(keyPair.d)})")
privKeyPEM = keyPair.exportKey()
print(privKeyPEM.decode('ascii'))


msg = bytes(str(input("Enter plain text: ")), 'utf-8')
encryptor = PKCS1_OAEP.new(pubKey)
encrypted = encryptor.encrypt(msg)
print("Encrypted:", binascii.hexlify(encrypted))

decryptor = PKCS1_OAEP.new(keyPair)
decrypted = decryptor.decrypt(encrypted)
print('Decrypted:', decrypted.decode('utf-8'))
```

```
from Crypto.PublicKey import RSA
from base64 import b64encode
from Crypto.Cipher import PKCS1_OAEP
import sys

msg = "hello..."

if (len(sys.argv)>1):
        msg=str(sys.argv[1])

key = RSA.generate(1024)

binPrivKey = key.exportKey('PEM')
binPubKey =  key.publickey().exportKey('PEM')

print ("====Private key===")
```

```
print (binPrivKey)
print ("====Public key===")
print (binPubKey)
privKeyObj = RSA.importKey(binPrivKey)
pubKeyObj =  RSA.importKey(binPubKey)
cipher = PKCS1_OAEP.new(pubKeyObj)
ciphertext = cipher.encrypt(msg.encode())
print ("====Ciphertext===")
print (b64encode(ciphertext))
cipher = PKCS1_OAEP.new(privKeyObj)
message = cipher.decrypt(ciphertext)
print ("====Decrypted===")
print ("Message:",message)
```

2. Mã Elgamal

pip install PyCryptoDomex

```
from elgamal.elgamal import Elgamal

m = b'Text'
print(m)
pb, pv = Elgamal.newkeys(128)
print(pb)
print(pv)
ct = Elgamal.encrypt(m, pb)
print(ct)
dd = Elgamal.decrypt(ct, pv)
print(dd)
print()
```