Άσκηση 3 Dai16067 Κρυπτογραφία

```
In [48]: # επίθεση επιλεγμένου απλού κειμένου - CPA
         n=26
         Z26=IntegerModRing(n)

         plaintext='WE'
         ciphertext='MW'

         def affine_analysis(m,c):
             mList=str2lst(m)
             cList=str2lst(c)


             A = matrix(Z26, 2, 2, [mList[0], 1, mList[1], 1])

             b = vector(Z26, [cList[0], cList[1]])


             key=A.solve_right(b)

             return key

         key=affine_analysis(plaintext,ciphertext)
         print 'The key is:', key

         print("The ciphertext message: " + ciphertext + " is decrypted in --> " + affine_dec(ciphertext,int(key[0]),int(key[1])))
```

```
The key is: (24, 4)

---------------------------------------------------------------------------
ZeroDivisionError                         Traceback (most recent call last)
<ipython-input-48-1ce1ef5f07f7> in <module>()
     23 print 'The key is:', key
     24
---> 25 print("The ciphertext message: " + ciphertext + " is decrypted in --> " + affine_dec(ciphertext,int(key[Integer(0)]),in
t(key[Integer(1)])))

<ipython-input-47-4bb86c345877> in affine_dec(c, k1, k2)
     14
     15 def affine_dec(c,k1,k2):
---> 16     k1_inverse=inverse_mod(k1,Integer(26))
     17     ciphertextList = str2lst(c)
     18     plaintextList = [(k1_inverse*(x-k2))%Integer(26) for x in ciphertextList]

/opt/sagemath-8.3/local/lib/python2.7/site-packages/sage/arith/misc.pyc in inverse_mod(a, m)
   1856         return a.inverse_mod(m)
   1857     except AttributeError:
-> 1858         return Integer(a).inverse_mod(m)
   1859
   1860 ######################################################

/opt/sagemath-8.3/local/lib/python2.7/site-packages/sage/rings/integer.pyx in sage.rings.integer.Integer.inverse_mod (build/cyt
honized/sage/rings/integer.c:41101)()
   6572         sig_off()
   6573         if r == 0:
-> 6574             raise ZeroDivisionError("Inverse does not exist.")
   6575         return ans
   6576

ZeroDivisionError: Inverse does not exist.
```

Τα συμπεράσματα που εξάγουμε είναι ότι δεν συμπεριφέρεται καταλλήλως σε όλες τις περιπτώσεις που σημαίνει ότι δεν έχει λύση το παραπάνω πρόβλημα καθώς δεν αντιστρέφεται πάντα και αυτό συμβαίνει γιατί το m δεν αντιστρέφεται πάντα!

Καθώς στο known plaintext attack το κείμενο το τελικό το γνωρίζει ο επιτιθέμενος ενώ στο CPA το διαλέγει. Το κλειδί 11,4 όντως αποκρυπτογραφεί το μήνυμα MW και αυτό φαίνεται στην παρακάτω εικόνα

In [23]:

```python
ctx = 'MW'

k1=11
k2=4

def str2lst(s):
    return [ord(x)-65 for x in s]

def lst2str(lst):
    return ''.join([chr(x+65) for x in lst])

def affine_dec(c,key1,key2):
    key1_inv=inverse_mod(key1,26)
    ct  = str2lst(c)
    ct2 = [((key1_inv*(x-key2))%26) for x in ct]
    pl  = lst2str(ct2)
    return pl

print("The ciphertext message: " + ctx + " becomes --> " + affine_dec(ctx,k1,k2))
```

```
The ciphertext message: MW becomes --> WE
```

In [27]:

```python
# A τρόπος με k1, k2 ακεραίους
message = 'WE'
ciphertext='MW'

phi_n=[k for k in range(1,26) if gcd(k,26)==1]

for i in phi_n:
    for j in range(1,26):
        if affine_dec(ciphertext,i,j)==message:
            print i, j, message
            break
```

```
11 4 WE
```

Απλά στο ΚΡΑ δεν ορίζεται ο αντίστροφος.

2. Μια βελτίωση που θα μπορούσαμε να κάνουμε είναι να εμφανίζει τον λόγω για τον οποίο δεν θα εμφανίσει ποτέ το σωστό αλγόριθμο και να αποκρύψει τα μηνύματα error για να είναι πιο φιλικό στον χρήση όπως φαίνεται παρακάτω:

```
In [7]: # επίθεση επιλεγμένου απλού κειμένου - CPA
n=26
Z26=IntegerModRing(n)

plaintext='WE'
ciphertext='MW'

def affine_analysis(m,c):

    mList= str2lst(m)
    cList= str2lst(c)


    A = matrix(Z26, 2, 2, [mList[0], 1, mList[1], 1])

    b = vector(Z26, [cList[0], cList[1]])


    key=A.solve_right(b)

    return key

key=affine_analysis(plaintext,ciphertext)
print 'The key is:', key
try:
    print("The ciphertext message: " + ciphertext + " is decrypted in --> " + affine_dec(ciphertext,int(key[0]),int(key[1])))
except ZeroDivisionError:
    print("αυτος ο αριθμος δεν μπορει να αντιστραφει και δεν μπορει να υπολογιστει, δοκιμαστε ξανα!")
```

```
The key is: (24, 4)
αυτο το κειμενο δεν μπορει να αντιστραφει και δεν μπορει να υπολογιστει, δοκιμαστε ξανα!
```

3. a) ο αριθμός των πιθανών κλειδιών θα μπορούσαν να είναι: 26(όσα τα γράμματα της ΑΒ) * 12 (οι αντιστρέψιμοι) *26(όσα τα ΑΒ απλά για τη μεταβλητή c) = 8.112

b) VIPE και K (3,17,8) έστω το γράμμα V, C1= (3*1+17*21+8) mod 26=E ara VIPE=EUMK c-d) έφτιαξα τον αλγόριθμο και ταυτόχρονά έλυσα και το επόμενο ερώτημα οπότε είναι και οι 2 απαντήσεις μαζί σε 1:

```
In [9]:
ctx = 'SMQ'
N=26
k1=5
k2=11
k3=2

def str2lst(s):
    return [ord(x)-65 for x in s]

def lst2str(lst):
    return ''.join([chr(x+65) for x in lst])

def affine_dec(c,key1,key2,key3,N):
    key2_inv=inverse_mod(key2,26)
    ct  = str2lst(c)
    ct2 = [((key2_inv*(x-key1*ct.index(x) -key3))%N) for x in ct]
    pl  = lst2str(ct2)
    return pl

print("The ciphertext message: " + ctx + " becomes --> " + affine_dec(ctx,k1,k2,k3,N))

The ciphertext message: SMQ becomes --> SRY
```

e) CPA: Ο ομοπαραλληλικός κρυπταλγόριθμος είναι ευάλωτος σε μια επίθεση επιλεγμένου απλού κειμένου (Chosen Plaintext Attack - CPA). Έχοντας προσωρινή πρόσβαση στον μηχανισμό κρυπτογράφησης αρκεί η επιλογή 3 γραμμάτων για τον προσδιορισμό του κλειδιού. Και με μια απλή αντικατάσταση σε 3 συναρτήσεις μπορεί πολύ ευκολά να βρεθεί το κλειδί. C1 = k1i + k2mi +k3 (mod26)   i ανήκει στο {1,2,3}

C2 = k1i + k2mi +k3 (mod26)

C3 = k1i + k2mi +k3 (mod26)

f) η ίδια λογική ισχύει και στο KPA απλά τώρα επειδή δεν επιλεγεί κείμενο ο επιτιθέμενος υπάρχει πάλι το πρόβλημα το αντιστρόφου οπότε πάλι μπορεί να μην υπολογιστεί το κλειδί. Χρειάζεται τουλάχιστον 3 γράμματα για να βρει το κλειδί.