Crypto-Doc

Release 1.0

CABOS Matthieu

Oct 22, 2021

CONTENTS:

CHAPTER

ONE

NUMERIC BASE RECURSIVE BUILDER ALGORITHM

1.1 Numeric Base Recursive Builder Algorithm

This is the Main Base Builder Recursive Generator Algorithm

1.1.1 Algorithm

This is the main Base Table Builder Algorithm. The algorithm is ruled by these following steps :

- **Init time** : I init Time variable using the time library from Python
- **First Step Base Table** : This is the first step of the builder, I build the basic table from the tablebase function
- **First level of recursivity** : I build the first level of recursivity in safe mode using recursive_build function
- **Full Recursive algorithm** : We get the full computation of the table via the recursive_build_sup_lvl method.
- **Time calculation** : Computation of necessary time for the construction of the full array

Returns list of list : A list of list containing all the string values representing the full generateed Base Table array

---

1.1.2 Source Code

```
def table():
        rec_level_h = [6,6,6,6,6,5,5,5,5,5,5,5,5,5,5,5,5,5,4,4,4,4,4,4,4,4,4]
        rec_level_m = [5,5,5,5,5,4,4,4,4,4,4,4,4,4,4,4,4,3,3,3,3,3,3,3,3,3,3]
        rec_level_l = [4,4,4,4,4,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3]
        table = []
        bases = []
        tmp   = ()
        ok    = 0
        mini   =11
        mytime = 0
        fini = 0
        finalt = 0
        initt  =time.time()

        # Construction algorithm using recursivity to build the nearest max_int bound␣
↪tables (never equal 4 000 000 000...)
        for i in range (mini,37):
                ind = 1
                ok  = 0
                bases.append(tablebase(i))
                table.append(recursive_build(bases[i-mini]))
                while(not ok):
                        tmp=recursive_build_sup_lvl(bases[i-mini],table[i-mini],ind)
                        table[i-mini]=tmp[0]
                        ind+=1
                        if(ind==rec_level_l[i-mini]):
                                ok=1
        return table
```

1.2 tablebase

```
def tablebase(base)
```

1.2.1 Algorithm

This method allow to build the first step Numeric Base Transposition Table. I use the Horner's scheme procedure to build the correct table independantly of the base index. I can build Base from 1->a to 1->z, mean Base11 to Base36.

Parameters	Type	Description
base	*int*	The base index to build

Returns list : The builded first step base table.

1.2.2 Source Code

```
res = []
letter = 'a'
letterbis = 'A'

for i in range(0,base):
        if(i<10 or (i<=10 and base <=10)):
                res.append(str(i))
        if(i>=10 and base >10 and base<37):
                res.append(letter)
                letter=chr(ord(letter)+1)

return res
```

1.3 recursive_build

```
def recursive_build(table_base)
```

1.3.1 Algorithm

This function recursively build a full Base Table from an existing one. You can pass the first step table as already builded recursive table.

Parameters	Type	Description
table_base	*list*	Base Table array as list

Returns **str list** : The recursively builded Base Table

1.3.2 Source Code

```
res = []
for i in table_base:
        for j in table_base:
                res.append(i+j)
return res
```

1.4 recursive_build_sup_lvl_safe_mode

```
def recursive_build_sup_lvl_safe_mode(current,indice)
```

1.4.1 Algorithm

The variable indeice correspond to the pow index of the current recursive build. The current argument contain the current Base Tale array as list. Using once again the Horner's scheme, we can build each sup level without be limited by internal constraints.

Parameters	Type	Description
current	*list*	The current table to treat
indice	*int*	The pow indice

Returns list : A list containing the next level builded Base Table

---

1.4.2 Source Code

```
res = []
for i in current:
        res.append(str(indice)+str(i))
return res
```

1.5 recursive_build_sup_lvl

```
def recursive_build_sup_lvl(table_base,current,lvl)
```

1.5.1 Algorithm

The recursive_build_sup_lvl method is used to manage recursivity of the algorithm. I mean i have wrote the iterative version of the recursive function. So you can easely use it and control it.

Parameters	Type	Description
table_base	*str list*	my first step Base table array as list
current	*str list*	my current Base table array as list
lvl	*int*	Define the level of recursivity

Returns (list,int) Tuple : The (Base Table builded, Index of depth) Couple of informations.

1.5.2 Source Code

```
res   = []
break_ind = 0
for i in table_base:
        try :
                res.extend(recursive_build_sup_lvl_safe_mode(current,i))
        except:
                break_ind=1
                break
return (res,break_ind)
```

CHAPTER TWO

RAPTOR CRYPTOGRAPHIC ALGORITHM V1

2.1 Description of Crypter

Welcom to Raptor cryptographic help

This following instructions give you the full light on the given cryptographic algorithm "Raptor". In a firts time I will explain the main algorithm rules. Each of the function used can be found on the full source code and have a dedicated help section.

Description of the Main Raptor's Cryptographic Algorithm

2.1.1 Algorithm

This is the main algorithm of the program. It allows from a system argv string to crypt it and get a string,key couple as result. We will use this following variables to make it work :

- **table2** : A list of list containing all the necessary Base table from Base 11 to Base 37
- **Basemin** : 2 as default, it means the minimum base index to generate
- **Basemax** : 37 as default, it means the maximum base index to generate
- **chaine** : The string chain to crypt as system argv argument
- **choice** : A choice variable to manage the main loop (continue or quit)
- **Range** : Define the range of values generate into the corresponding Numeric Base a the begining

The return of the algorithm is ruled by the fllowing variables:

- **testkey** : The final half key as key
- **raw_txt** : The final crypted strin as string.

The alorithm is ruled by the following steps :

- Generating the first step Base table for each necessary numeric base via the function table and splitTable
- **Recursive build of the full Base table since the first step table using functions :**
  - rec_table_construct_lvl1 : It draw the 'zero theorem' of Table construction since the first step. Must be considered as te first loop of recursive builder algorithm
  - rec_manage : It draw the full Base Table using recursive loop
- Instanciation of the local varables to manipulate the algorithm
- I crypt the string using the crypt_procedure function. The return is a couple (crypt text / key) wich allow to decrypt it.
- **The crypt_final method allow us to organise the crypt list into interpretables results. We store results in variables:**
  - **raw_txt** : Contains the raw crypted text as string
  - **testkey** : Contains the half key as str(int)

This algorithm is stable in his domain and must be used on it. Please not to try bigger data slice and automate it via shell script if necessary. It should be used as a data crypter using a top level slicer and manager (from the shell script as exemple).

See source below to more explanation.

2.1.2 Source Code

```
import sys
import math as m

represent=''
table2 = []
dic = {}
main_dic={}
choice = ' '
chaine=''
choice=''
try : chaine=sys.argv[1]
except : print("")

if(len(sys.argv)!=4):
        Basemin = 2
        Basemax = 37
        Range   = 36**2
else :
        Basemin = int(sys.argv[1])
        Basemax = int(sys.argv[2])
        Range   = int(sys.argv[3])

if(Basemin<2 or Basemax>37):
        print("Affichage impossible veuillez selectionner une plage de valeure contenue␣
↪dans [2,36]")
        exit(0)

maxi=Basemax-Basemin

for i in range(Basemin,Basemax):
        table2.append(table(i,0,Range,1))

for i in range (0,len(table2)):
        table2[i]=splitTable(table2[i])

for j in range (0,len(table2)):
        table2[j]=rec_table_construct_lvl1(table2[j],j+2,1,0)
        for k in range(0,j+2):
                table2[j][k]=(str(0)+table2[j][k])
table2=rec_manage(table2)
testc=[]
testk=[]
while(choice!='q'):
        chaine=''
        res = ()
        testc[:]=[]
        testk[:]=[]
        raw_txt=''
        testkey=''
        while(len(chaine)>=29 or len(chaine)==0):
                chaine=input("Veuillez entrer une chaine <29 : \n")
```

(continues on next page)

(continued from previous page)

```
        res=crypt_procedure(chaine,table2)
        testc = res[0]
        testk = res[1]
        for i in range(0,len(testk)):
                testkey+=str(testk[i])
        raw_txt = crypt_final(res)
        print("------------------------------------------------------------------------
↪---------")
        print("Chaine cryptée : \n")
        print(raw_txt)
        print("------------------------------------------------------------------------
↪---------")
        print("Clé unique : \n")
        print(testkey)
        print("------------------------------------------------------------------------
↪---------")
        clean_txt = decrypt_procedure(raw_txt,testk,table2)
        print("Chaine décryptée : \n")
        print(clean_txt)
        choice=input("c)ontinuer ou q)uitter?")
```

2.2 Description of De-crypter

Description of the Main Raptor's Cryptographic Algorithm

2.2.1 Algorithm

This is the main solver algorithm program. It allow us to decrypt datas slices crypted with the version 1 of the Raptor Cryptographic Algorithm. To solve I need thse following variables :

- **chaine** : The input crypted string storage
- **Basemin** : The minimum Base index
- **Basemax** : The maximum Base index
- **table2** : The list of list containing the Base Table
- **finalkey** : The key of the algorithm, the decrypting process absolutely need this key.

The solving procedure is ruled by the following steps:

- **Generating the Base Table** and store it into my table2 variable
- **Getting inputs** known as crypted string and his associated key.
- **Decrypting process** using the decrypt_procedure method (see documentation)
- **Store and return** the results of decrypting process

2.2.2 Source Code

```
import sys
import math as m

represent=''
table2 = []
dic = {}
main_dic={}
choice = ' '
chaine=''
print("-----------------------------------------------------------------------------
↪")

if(len(sys.argv)!=4):
        Basemin = 2
        Basemax = 37
        Range   = 36**2
else :
        Basemin = int(sys.argv[1])
        Basemax = int(sys.argv[2])
        Range   = int(sys.argv[3])
```

(continues on next page)

(continued from previous page)

```
if(Basemin<2 or Basemax>37):
        print("Affichage impossible veuillez selectionner une plage de valeure contenue␣
↪dans [2,36]")
        exit(0)

maxi=Basemax-Basemin

for i in range(Basemin,Basemax):
        table2.append(table(i,0,Range,1))

for i in range (0,len(table2)):
        table2[i]=splitTable(table2[i])

for j in range (0,len(table2)):
        table2[j]=rec_table_construct_lvl1(table2[j],j+2,1,0)
        for k in range(0,j+2):
                table2[j][k]=(str(0)+table2[j][k])
table2  = rec_manage(table2)

finalke=[]
while(choice!='q'):
        finalke[:]=[]
        finalkey=''
        decrypt=''
        chaine=input("Veuillez entrer la chaine cryptée : \n")
        print("-------------------------------------------------------------------------
↪--------")
        finalkey= input("Veuillez saisir la clé : \n")
        finalke = miam(finalkey)
        decrypt = decrypt_procedure(chaine,finalke,table2)
        print("-------------------------------------------------------------------------
↪--------")
        print("Chaine decryptée : ")
        print(decrypt)
        choice=input("c)ontinuer ou q)uitter ?")
```

2.3 reverse

```
def reverse(s)
```

2.3.1 Algorithm

A function to reverse a string as argument.

Parameter	Type	Description
s	*String*	The string to reverse

Returns str : The reversed string

2.3.2 Source Code

```
str= ""
for i in s:
        str=i+str
return str
```

2.4 splitTable

```
def splitTable(table)
```

2.4.1 Algorithm

Split a string as array from the given separator.

Parameters	Type	Description
table	*string*	The list to split

Returns **list** : The splitted list

2.4.2 Source Code

```
local_list=table.split('\n')
res_list=[]
for i in range (0,len(local_list)):
        res_list.append(local_list[i])
return res_list
```

2.5 table

```
def table(base,debut,fin,inc)
```

2.5.1 Algorithm

Base table recursive builder. The generated Base table array is defined via :

- **base** : Define the base to begin the table
- **debut** : Define the first value of Base table
- **fin** : Define the last value of Base table
- **inc** : Define the incrementation step

Parameters	Type	Description
base	*int*	The first base of the table
debut	*int*	The first value of the table in the given base
fin	*int*	The last value of the table in the given base
inc	*int*	The value of incrementation step

Returns Str : A string containing all the base generated representing the array (see conversion later)

2.5.2 Source Code

```
represent=''
letter='a'
powIndex=0
count=0
if(fin>10*base):
        fin=10*base
for i in range(debut,fin):
        current=i
        if(i<base):
                if(i<10):
                        represent+=str(i)
                else:
                        represent+=letter
                        letter=chr(ord(letter)+1)
                if(i==base-1):
                        letter='a'
        else:
                tmp=''
                while(current/base!=0):
                        count=powIndex*10*base
                        if(not current%(10*base)):
                                                powIndex+=1
```

(continues on next page)

(continued from previous page)

```
                        if(base<10):
                                tmp+=str(current%base)
                        else:
                                if(current%base<10):
                                        tmp+=str(current%base)
                                else:
                                        tmp+=letter
                                        if(count==0):
                                                letter=chr(ord(letter)+1)
                                        else:
                                                count-=1
                                        if(current%base==base-1):
                                                letter='a'
                        current=int(current/base)
                represent+=reverse(tmp)
        represent+="\n"
return represent
```

2.6 rec_table_construct_lvl1

```
def rec_table_construct_lvl1(table,base,powindex,last)
```

2.6.1 Algorithm

Recursive Construction method from the Base table. The recursive algorithm permit to edit much larger array from existing original base table. Ths algorithm must be used as the init loop of the final recursive method (see rec_manage method)

Parameters	Type	Description
table	*list*	The Base table array
base	*int*	The current numeric base as integer
powindex	*int*	The pow index as integer
last	*int*	unused

Returns **list** : The Recursively builded Base table as list

---

2.6.2 Source Code

```
lettrebase=table[10:base]
if(powindex == 1):
        del table[10*base]
res=table[:]
for i in range (len(table)-1,base**2-1):
        if(i%base==(base-1) and i!=len(table)-1):
                powindex+=1
        res.append(lettrebase[powindex-1]+str(table[(i-len(table)+1)%base]))
return res
```

2.7 rec_table_construct_final

```
def rec_table_construct_final(table,base,lvl)
```

2.7.1 Algorithm

Recursive Construction method from the Base table. The recursive algorithm manage array building since 2 levels of recursive construction. => Do not use for the first recursive building loop

Parameters	Type	Description
table	*list*	The first recursive level builded Base table
base	*int*	The base to treat as integer
lvl	*int*	The level of recursivity in construction

Returns **list** : The fully specified level recursivity builded Base table

2.7.2 Source Code

```
res=[]
basetable=table[0:base]
for i in range(0,len(basetable)):
        basetable[i]=basetable[i][lvl:]
for eat in basetable:
        for this in table:
                res.append(eat+this)
return res
```

2.8 rec_manage

```
def rec_manage(table)
```

2.8.1 Algorithm

A recursivity manager to build properly the base table. It must be used to map the numeric values into base values. This method allow contruction of hundreds of thousand values table

Parameters	Type	Description
table	*list*	The initial Base table to complete

Returns list : The fully builded Base table

2.8.2 Source Code

```
for i in range(9,len(table)):
        table2=table[i][:]
        table2=rec_table_construct_lvl1(table2,i+2,1,0)
        table2=rec_table_construct_final(table2,i+2,1)
        table2=rec_table_construct_final(table2,i+2,2)
        if(i<20):
                table2=rec_table_construct_final(table2,i+2,3)
        table[i]=table2[:]
return table
```

2.9 ascii_to_int

```
def ascii_to_int(chaine)
```

2.9.1 Algorithm

Utils method : ascii to integer converter.

Parameters	Type	Description
chaine	*str*	The string to convert

Returns **list** : A list containing all integers values since ASCII.

2.9.2 Source Code

```
res = []
for letter in chaine:
        res.append(ord(letter))
return res
```

2.10 int_to_ascii

```
def int_to_ascii(crypt)
```

2.10.1 Algorithm

Utils method : integer to ascii converter.

Description	Type	Description
crypt	*int*	The int list to convert

Returns **str** : The converted ASCII string since int list.

2.10.2 Source Code

```
res = ''
for i in range (0,len(crypt)):
        res+=chr(crypt[i])
return res
```

2.11 cryptChaine

```
def cryptChaine(to_crypt,table,base)
```

2.11.1 Algorithm

The simple method to crypt an ascii string as integer list.

Parameters	Type	Description
to_crypt	*int list*	The converted int list since an ascii string
table	*list of list*	An array containing all fully builded Base Table
base	*int*	Define the Base index

Returns str list : A string list containing all the base crypted values, Must be used as a crypted list.

2.11.2 Source Code

```
res = []
for i in range(0,len(to_crypt)):
        res.append(table[base][to_crypt[i]])
return res
```

2.12 local_table_dico

```
def local_table_dico(table2,base,rangeB)
```

2.12.1 Algorithm

Utils method : A method to convert a Base table to Python dictionnary

Parameters	Type	Description
table2	*list of list*	An array containing all the fully builded Base table
base	*int*	Define the Base index
rangeB	*int*	Define the max step of incrementation

Returns **Dictionnary** : A dictionnary representing the specified Base table

2.12.2 Source Code

```
str_base={}
res = {}
if(rangeB>base**2):
        rangeB=base**2
for i in range (0,rangeB):
        str_base[i]=table2[base][i]
return str_base
```

2.13 limit_range

```
def limit_range(Range,base)
```

2.13.1 Algorithm

Utils method : A method to limit the Base range

Parameters	Type	Description
Range	*int*	The range as a limit
base	*int*	The current Base index

Returns int : The limited by range res.

2.13.2 Source Code

```
res=0
if(Range>base**2):
        res=base**2
else:
        res=Range
return res
```

2.14 base_key

```
def base_key(int_chaine)
```

2.14.1 Algorithm

This is the key builder.

Parameters	Type	Description
int_chaine	*int list*	The base index list as a starting builder for key

Returns int list : the builded key from index base list.

2.14.2 Source Code

```
res=[]
for i in range (0,len(int_chaine)):
        tmp=((int_chaine[i]*int_chaine[len(int_chaine)-i-1]+10)%36)
        if(tmp<10):
                tmp+=10
        res.append(tmp)
return res
```

2.15 vec_poids

```
def vec_poids(int_chaine)
```

2.15.1 Algorithm

Compute the vectorial cumulated weight of the list.

Parameters	Type	Description
int_chaine	*int list*	The integer list to treat

Returns int list : The computed accumulated weigth integer list

---

2.15.2 Source Code

```
res = []
res.append(int_chaine[0])
for i in range(1,len(int_chaine)):
        res.append(res[i-1]+int_chaine[i])
return res
```

2.16 vec_1_poids

```
def vec_1_poids(vec_poids)
```

2.16.1 Algorithm

Compute the inverse of the vectorial cumulated weigth computation.

Parameters	Type	Description
vec_poids	*int list*	The weigth as an integer list

Returns int list : The computed list containing the inverse operation of vec_poids method

2.16.2 Source Code

```
res=[]
for i in range (0,len(vec_poids)):
        res.append(1/vec_poids[i])
return res
```

2.17 equa_2_nd

```
def equa_2_nd(a,b,c)
```

2.17.1 Algorithm

Utils : An 2nd order equation solver

Parameters	Type	Description
a	*int / float*	The a coefficient
b	*int / float*	The b coefficient
c	*int / float*	The c coefficient

Returns int / float : The solved equation positive root

2.17.2 Source Code

```
res = 0
racine1 = 0.0
racine2 = 0.0
delta = b**2-4*a*c
if(delta>0):
        racine1 = (-b+m.sqrt(delta))/2*a
        racine2 = (-b-m.sqrt(delta))/2*a
if(racine1>0):
        res = int(racine1)
else:
        res = int(racine2)
return res
```

2.18 multlist

```
def multlist(a,b)
```

2.18.1 Algorithm

Utils : A point by point list multiplier

Parameters	Type	Description
a	*int/float list*	The list to multiply
b	*int/float list*	The list to multiply

Returns int / float list : The computed point by point multiplication

2.18.2 Source Code

```
res = []
if(len(a)!=len(b)):
        return []
else:
        for i in range(0,len(a)):
                res.append(a[i]*b[i])
return res
```

2.19 transpose_base

```
def transpose_base(liste,key,table)
```

2.19.1 Algorithm

A method to transpose an integer list to the corresponding key's base index => The result will be a succession of transposed values from differents integers to differents base

Parameters	Type	Description
liste	*list*	The integer converted since ASCII list
key	*list*	The Base index list as key
table	*list*	The full Base Table recursively builded

Returns **str list**: The crypted list as String list

2.19.2 Source Code

```
res = []
if(len(liste)!=len(key)):
        return []
else :
        for i in range (0,len(liste)):
                if(key[i]==10):
                        res.append(liste[i])
                else:
                        res.append(table[key[i]-2][liste[i]])
return res
```

2.20 inv_transpose_base

```
def inv_transpose_base(liste,key,table)
```

2.20.1 Algorithm

The inverse method to decrypt a str list of base transposed values

Parameters	Type	Description
liste	*str list*	The crypted list as String list
key	*int list*	The Base index list as key
table	*int list*	The full Base table recursively builded

Returns **int list** : The decrypted list as integers

2.20.2 Source Code

```
res = []
if(len(liste)!=len(key)):
        return []
else:
        for i in range(0,len(liste)):
                if(key[i]==10):
                        res.append(int(liste[i]))
                else:
                        res.append(int(table[key[i]-2].index(liste[i])))
return res
```

2.21 crypt_procedure

```
def crypt_procedure(chaine,table)
```

2.21.1 Algorithm

The crypter manager to orchestrate the crypting procedure. It works from these steps:

- *We convert the given ascii string as integer list*
- *We compute the Base index list as key from the converted integer list*
- *We build the second part of the key since the mirror of the Base index list*
- *We compute the cumulated weight of the integer list*
- *We compute the point by point multiplication between cumulated weigth list and original integer list*
- *We transpose the multiplied list into the given specified Base from the key*
- *We associate the crypted strin to the key as return*

Parameters	Type	Description
chaine	*string*	The string to crypt
table	*list of list*	The Base Table recursively builded

Returns list tuple : The couple crypted string and key as result. It permits to decrypt any message.

2.21.2 Source Code

```
int_chaine = ascii_to_int(chaine)
base_keyy  = base_key(int_chaine)
if(len(base_keyy)%2==0):
        key=base_keyy[0:int(len(base_keyy)/2)]
else:
        key=base_keyy[0:int((len(base_keyy)/2)+1)]
vec_poid   = vec_poids(int_chaine)
crypt_lst  = multlist(int_chaine,vec_poid)
crypt_lst  = transpose_base(crypt_lst,base_keyy,table)
return(crypt_lst,key)
```

2.22 cyclik_ascii

```
def cyclik_ascii(current)
```

2.22.1 Algorithm

Compute a cyclik ascii separators into ponctuation signs

Parameters	Type	Description
current	*str*	The current poncuation separator

Returns **str** : The following separator from the defined 'sep' Set.

2.22.2 Source Code

```
sep=['!','"','#','$','%','&','(',')','*','+',',','-','.','/']
tmp=((sep.index(current)+1)%13)
res =sep[tmp]
return res
```

2.23 crypt_final

```
def crypt_final(tuple)
```

2.23.1 Algorithm

The layout procedure to organise crypting results.

Parameters	Type	Description
tuple	*tuple*	List couple representing the crypted strin and the associated key

Returns str : The crypted list as a string with correct separators

2.23.2 Source Code

```
res = ''
sep = '!'
crypt=tuple[0]
key=tuple[1]
for i in range (0,len(crypt)):
        res+=sep+str(crypt[i])
        sep=cyclik_ascii(sep)
return res
```

2.24 slurp

```
def slurp(chaine)
```

2.24.1 Algorithm

This method allow us to rebuild a str list of crypted terms using separators set.

Parameters	Type	Description
chaine	*str*	The raw string crypted message

Returns str list : The list of crypted terms rebuilded from the raw string

2.24.2 Source Code

```
tmp=''
res = []
sep=['!','"','#','$','%','&','(',')','*','+',',','-','.','/']
for elem in chaine:
        if(not elem in sep):
                tmp+=str(elem)
        else :
                res.append(tmp)
                tmp=''
        if(elem==''):
                break
res=res[1:]
res.append(tmp)
return res
```

2.25 resolve

```
def resolve(liste)
```

2.25.1 Algorithm

This method compute the chained 2nd order equations to solve the numeric suit. It permit us to get the ASCII values as a list. To solve the system you have to instance the solver with the square root of term 0. Once theorem zero done, you will apply the equation solver with square root of the 0-term as b, a as 1 and c as -following term. The algorithm sort the roots and take only positives ones.

Parameters	Type	Description
liste	*int list*	The computed multiplied list to solve

Returns **int list** : A list containing solved terms.

2.25.2 Source Code

```
res = []
x = 0
tmp2 = 0
res.append(int(m.sqrt(liste[0])))
tmp=res[0]
for i in range (1,len(liste)):
        tmp2 = equa_2_nd(1,-tmp,-liste[i])
        x=tmp2-tmp
        res.append(int(x))
        tmp=tmp2
return res
```

2.26 decrypt_procedure

```
def decrypt_procedure(chaine,key,table)
```

2.26.1 Algorithm

This method manage the decrypting procedure. It is ruled by the following steps :

- *Build the full key since the key argument*
- *Split the string since separators via slurp method*
- *Apply the inv_tranpose_base method to get the uncrypted terms*
- *Solve the cumulated multiplued weigth with the equation solver*
- *Convert the int list as result to ASCII chain*

Parameters	Type	Description
chaine	*str*	The raw crypted text as string
key	*int list*	The half key as int list
table	*list of list*	The Base Table array

Returns str : The uncrypted text.

2.26.2 Source Code

```
res = ''
base=key[:]
tmp = []
key.reverse()
tmp = key[:]
to_find = []
to_find=slurp(chaine)
if(len(to_find)%2==0):
        base+=tmp[0:len(key)]
else:
        base+=tmp[1:len(key)]
tmp_liste=inv_transpose_base(to_find,base,table)
int_liste=resolve(tmp_liste)
res = int_to_ascii(int_liste)
return res
```

CHAPTER

THREE

RAPTOR CRYPTOGRAPHIC ALGORITHM V2

3.1 Description of Crypter

Welcome to Raptor cryptographic help

This following instructions give you the full light on the given cryptographic algorithm "Raptor". In a firts time I will explain the main algorithm rules. Each of the function used can be found on the full source code and have a dedicated help section.

Description of the Main Raptor's Cryptographic Algorithm

3.1.1 Algorithm

This is the main algorithm of the program. It allows from a system argv string to crypt it and get a string,key couple as result. We will use this following variables to make it work :

- **table2** : A list of list containing all the necessary Base table from Base 11 to Base 37
- **Basemin** : 2 as default, it means the minimum base index to generate
- **Basemax** : 37 as default, it means the maximum base index to generate
- **chaine** : The string chain to crypt as system argv argument
- **choice** : A choice variable to manage the main loop (continue or quit)
- **Range** : Define the range of values generate into the corresponding Numeric Base a the begining

The return of the algorithm is ruled by the following variables:

- **testkey** : The final half key as key
- **raw_txt** : The final crypted strin as string.

The alorithm is ruled by the following steps :

- **Generating the first step Base table** for each necessary numeric base via the function table and splitTable
- **Recursive build of the full Base table** since the first step table using functions :
  - **rec_table_construct_lvl1** : It draw the 'zero theorem' of Table construction since the first step. Must be considered as te first loop of recursive builder algorithm
  - **rec_manage** : It draw the full Base Table using recursive loop
- **Initialization** : Instanciation of the local varables to manipulate the algorithm
- **Split** : I crypt the data string as input using slices of the string vector. Using a loop, I will crypt each slices independantly from each others. It permits us to have a full crypted string more complex than the first version of algorithm
- **Crypting Slices** : Once each slices properly cutted, we have to crypt each of them using the crypt_procedure automated on a loop coursing each of them.
- **Manage Slices** : The crypted slices are managed via a second level separators set wich define a second level of crypting tree. In fact each term of a slice is using a first level of separators, it give a one-level tree. The second level permit to complexify the full algorithm result.
- **Rebuild results** : Finally, the crypt_procedure function is used to associate each crypted slice to his key and draw a correct interpretated result as list of couple (crypted string/integer key)
- **Return results** : The couple full result rebuilded from slices couple is organized from the second level separators to draw a 2-level tree

This algorithm is stable in his domain and must be used on it. Please not to try bigger data slice and automate it via shell script if necessary. It should be used as a data crypter using a top level slicer and manager (from the shell script as exemple).

See source below to more explanation.

3.1.2 Source Code

```
import sys
import math as m
import random as r

represent=''
table2 = []
dic = {}
main_dic={}
choice = ' '
chaine=''

#system check routine
if(len(sys.argv)!=4):
        Basemin = 2
        Basemax = 37
        Range   = 36**2
else :
        Basemin = int(sys.argv[1])
        Basemax = int(sys.argv[2])
        Range   = int(sys.argv[3])

if(Basemin<2 or Basemax>37):
        print("Affichage impossible veuillez selectionner une plage de valeure contenue␣
↪dans [2,36]")
        exit(0)

#init routine
maxi=Basemax-Basemin

for i in range(Basemin,Basemax):
        table2.append(table(i,0,Range,1))

for i in range (0,len(table2)):
        table2[i]=splitTable(table2[i])

for j in range (0,len(table2)):
        table2[j]=rec_table_construct_lvl1(table2[j],j+2,1,0)
        for k in range(0,j+2):
                table2[j][k]=(str(0)+table2[j][k])
table2=rec_manage(table2)

#second level local declaration
long_chaine = []
```

(continues on next page)

(continued from previous page)

```
long_crypt  = []
longi=0
seuil = 20
choice = ''
userchoice=1
#definition of sets
sep=['!','"','#','$','%','&','(',')','*','+',',','-','.','/']
sep_lvl2=[":",";","<","=",">","?","@"]
long_chaine   = []
long_crypt    = []
testc         = []
testk         = []
int_chaine    = []
lvl2_key_miam = []

#main algorithm
while(choice!='q'):
        # init_all()
        current_sep_lvl2 =  ":"
        long_chaine[:]   = []
        long_crypt[:]    = []
        testc[:]         = []
        testk[:]         = []
        int_chaine[:]    = []
        lvl2_key_miam[:] = []
        testkey=''
        raw_txt=''
        clean_txt = ''
        longi = 0

        res = ()
        if(userchoice):
                chaine = ''
                chaine=input("Veuillez entrer la chaine à crypter : ")
        if(len(chaine)>=20):
                long_chaine = split(chaine,seuil)
                longi+=1
        if(not longi):
                res=crypt_procedure(chaine,table2)
        else :
                for i in range(0,len(long_chaine)):
                        long_crypt.append(crypt_procedure(long_chaine[i],table2))
        if(not longi):
                testc = res[0]
                testk = res[1]
        else :
                for i in range (0,len(long_crypt)):
                        for j in range(0,len(long_crypt[i][0])):
                                testc.append(str(long_crypt[i][0][j]))
                        for k in range(0,len(long_crypt[i][1])):
                                testk.append(str(long_crypt[i][1][k]))
                        current_sep_lvl2=cyclik_ascii_lvl2(current_sep_lvl2)
```

(continues on next page)

(continued from previous page)

```
                        testc[-1]+=current_sep_lvl2
                        testk[-1]+=current_sep_lvl2

        int_chaine=(ascii_to_int(chaine))
        for i in range(0,len(testk)):
                testkey+=str(testk[i])

        if(not longi):
                raw_txt = crypt_final(res,int_chaine)
        else:
                raw_txt += crypt_final_long(testc,int_chaine)

        print("Chaine cryptée : \n")
        print(raw_txt)
        print("Clé unique : \n")
        print(testkey)

        if(not longi):
                clean_txt = decrypt_procedure(raw_txt,testk,table2)
        else:
                lvl2_liste = []
                lvl2_key   = []
                lvl2_liste = slurp2(raw_txt)
                lvl2_key   = slurp2(testkey)
                lvl2_key_miam = []
                for i in range (0,len(lvl2_key)):
                        lvl2_key_miam.append(miam(lvl2_key[i]))
                for i in range (0,len(lvl2_liste)-1):
                        clean_txt+= decrypt_procedure(lvl2_liste[i],lvl2_key_miam[i],
↪table2)
        print("Chaine décryptée : \n")
        print(clean_txt)
        choice=input("c)ontinuer ou q)uitter")
        if(choice!='q'):
                userchoice+=1
```

3.2 Description of De-crypter

Description of the Main Raptor's Cryptographic Algorithm

3.2.1 Algorithm

This is the main solver algorithm program. It allow us to decrypt datas slices crypted with the version 1 of the Raptor Cryptographic Algorithm. To solve I need thse following variables :

- **raw_txt** : The input crypted string storage
- **Basemin** : The minimum Base index
- **Basemax** : The maximum Base index
- **table2** : The list of list containing the Base Table
- **testkey** : The key of the algorithm, the decrypting process absolutely need this key.

The solving procedure is ruled by the following steps:

- **Generating the Base Table** and store it into my table2 variable
- **Getting inputs** known as crypted string and his associated key.
- **Decrypting process** using the decrypt_procedure method (see documentation)
- **Store and return** the results of decrypting process

3.2.2 Source Code

```
import sys
import math as m
import random as r

represent=''
table2 = []
dic = {}
main_dic={}
choice = ' '
chaine=''

#system check routine
if(len(sys.argv)!=4):
        Basemin = 2
        Basemax = 37
        Range   = 36**2
else :
        Basemin = int(sys.argv[1])
        Basemax = int(sys.argv[2])
        Range   = int(sys.argv[3])
```

(continues on next page)

(continued from previous page)

```
if(Basemin<2 or Basemax>37):
        print("Affichage impossible veuillez selectionner une plage de valeure contenue␣
↪dans [2,36]")
        exit(0)

#init routine
maxi=Basemax-Basemin

for i in range(Basemin,Basemax):
        table2.append(table(i,0,Range,1))

for i in range (0,len(table2)):
        table2[i]=splitTable(table2[i])

for j in range (0,len(table2)):
        table2[j]=rec_table_construct_lvl1(table2[j],j+2,1,0)
        for k in range(0,j+2):
                table2[j][k]=(str(0)+table2[j][k])
table2=rec_manage(table2)

#second level local declaration
long_chaine = []
long_crypt  = []
longi=0
seuil = 20
choice = ''
userchoice=0
#definition of sets
sep=['!','"','#','$','%','&','(',')','*','+',',','-','.','/']
sep_lvl2=[":",";","<","=",">","?","@"]
long_chaine   = []
long_crypt    = []
testc         = []
testk         = []
int_chaine    = []
lvl2_key_miam = []

#main algorithm
while(choice!='q'):
        # init_all()
        current_sep_lvl2 =  ":"
        long_chaine[:]   = []
        long_crypt[:]    = []
        testc[:]         = []
        testk[:]         = []
        int_chaine[:]    = []
        lvl2_key_miam[:] = []
        testkey=''
        raw_txt=''
        clean_txt = ''
        longi = 0
```

(continues on next page)

(continued from previous page)

```
        res = ()
        raw_txt=input('Chaine cryptée : ')
        testkey=input('Clé unique : ')
        if(len(raw_txt)>=120):
                longi=1
        if(not longi):
                testkey=miam(testkey)
                clean_txt = decrypt_procedure(raw_txt,testkey,table2)
        else:
                lvl2_liste = []
                lvl2_key   = []
                lvl2_liste = slurp2(raw_txt)
                lvl2_key   = slurp2(testkey)
                lvl2_key_miam = []
                for i in range (0,len(lvl2_key)):
                        lvl2_key_miam.append(miam(lvl2_key[i]))
                for i in range (0,len(lvl2_liste)-1):
                        clean_txt+= decrypt_procedure(lvl2_liste[i],lvl2_key_miam[i],
↪table2)
        print("Chaine décryptée : \n")
        print(clean_txt)
        choice=input("c)ontinuer ou q)uitter")
        if(choice!='q'):
                userchoice+=1
```

3.3 reverse

```
def reverse(s)
```

3.3.1 Algorithm

A function to reverse a string as argument.

Parameter	Type	Description
s	*String*	The string to reverse

Returns **str** : The reversed string

3.3.2 Source Code

```
str= ""
for i in s:
        str=i+str
return str
```

3.4 splitTable

```
def splitTable(table)
```

3.4.1 Algorithm

Split a string as array from the given separator.

Parameters	Type	Description
table	*string*	The list to split

Returns **list** : The splitted list

3.4.2 Source Code

```
local_list=table.split('\n')
res_list=[]
for i in range (0,len(local_list)):
        res_list.append(local_list[i])
return res_list
```

3.5 table

```
def table()
```

3.5.1 Algorithm

Base table recursive builder. The generated Base table array is defined via :

- **base** : Define the base to begin the table
- **debut** : Define the first value of Base table
- **fin** : Define the last value of Base table
- **inc** : Define the incrementation step

Parameters	Type	Description
base	*int*	The first base of the table
debut	*int*	The first value of the table in the given base
fin	*int*	The last value of the table in the given base
inc	*int*	The value of incrementation step

Returns Str : A string containing all the base generated representing the array (see conversion later)

3.5.2 Source Code

```
represent=''
letter='a'
powIndex=0
count=0
if(fin>10*base):
        fin=10*base
for i in range(debut,fin):
        current=i
        if(i<base):
                if(i<10):
                        represent+=str(i)
                else:
                        represent+=letter
                        letter=chr(ord(letter)+1)
                if(i==base-1):
                        letter='a'
        else:
                tmp=''
                while(current/base!=0):
                        count=powIndex*10*base
                        if(not current%(10*base)):
                                                powIndex+=1
```

(continues on next page)

(continued from previous page)

```
                        if(base<10):
                                tmp+=str(current%base)
                        else:
                                if(current%base<10):
                                        tmp+=str(current%base)
                                else:
                                        tmp+=letter
                                        if(count==0):
                                                letter=chr(ord(letter)+1)
                                        else:
                                                count-=1
                                        if(current%base==base-1):
                                                letter='a'
                        current=int(current/base)
                represent+=reverse(tmp)
        represent+="\n"
return represent
```

3.6 rec_table_construct_lvl1

```
def rec_table_construct_lvl1()
```

3.6.1 Algorithm

Recursive Construction method from the Base table. The recursive algorithm permit to edit much larger array from existing original base table. Ths algorithm must be used as the init loop of the final recursive method (see rec_manage method)

Parameters	Type	Description
table	*list*	The Base table array
base	*int*	The current numeric base as integer
powindex	*int*	The pow index as integer
last	*int*	unused

Returns **list** : The Recursively builded Base table as list

3.6.2 Source Code

```
lettrebase=table[10:base]
if(powindex == 1):
        del table[10*base]
res=table[:]
for i in range (len(table)-1,base**2-1):
        if(i%base==(base-1) and i!=len(table)-1):
                powindex+=1
        res.append(lettrebase[powindex-1]+str(table[(i-len(table)+1)%base]))
return res
```

3.7 rec_table_construct_final

```
def rec_table_construct_final(table,base,lvl)
```

3.7.1 Algorithm

Recursive Construction method from the Base table. The recursive algorithm manage array building since 2 levels of recursive construction. => Do not use for the first recursive building loop

Parameters	Type	Description
table	*list*	The first recursive level builded Base table
base	*int*	The base to treat as integer
lvl	*int*	The level of recursivity in construction

Returns list : The fully specified level recursivity builded Base table

3.7.2 Source Code

```
res=[]
basetable=table[0:base]
for i in range(0,len(basetable)):
        basetable[i]=basetable[i][lvl:]
for eat in basetable:
        for this in table:
                res.append(eat+this)
return res
```

3.8 rec_manage

```
def rec_manage(table)
```

3.8.1 Algorithm

A recursivity manager to build properly the base table. It must be used to map the numeric values into base values. This method allow contruction of hundreds of thousand values table

Parameters	Type	Description
table	*list*	The initial Base table to complete

Returns **list** : The fully builded Base table

3.8.2 Source Code

```
j=0
for i in range(9,len(table)):
        table2=table[i][:]
        table2=rec_table_construct_lvl1(table2,i+2,1,0)
        table2=rec_table_construct_final(table2,i+2,1)
        table2=rec_table_construct_final(table2,i+2,2)
        table[i]=table2[:]
for i in range (9,18):
        j=3
        table2=table[i][:]
        while(len(table2)<1000000):
                table2=rec_table_construct_final(table2,i+2,j)
                j+=1
        table[i]=table2[:]
return table
```

3.9 ascii_to_int

```
def ascii_to_int(chaine)
```

3.9.1 Algorithm

Utils method : ascii to integer converter.

Parameters	Type	Description
chaine	*str*	The string to convert

Returns **list** : A list containing all integers values since ASCII.

3.9.2 Source Code

```
res = []
for letter in chaine:
        res.append(ord(letter))
return res
```

3.10 int_to_ascii

```
def int_to_ascii(crypt)
```

3.10.1 Algorithm

Utils method : integer to ascii converter.

Description	Type	Description
crypt	*int list*	The int list to convert

Returns **str** : The converted ASCII string since int list.

3.10.2 Source Code

```
res = ''
for i in range (0,len(crypt)):
        res+=chr(crypt[i])
return res
```

3.11 cryptChaine

```
def cryptChaine(to_crypt,table,base)
```

3.11.1 Algorithm

The simple method to crypt an ascii string as integer list.

Parameters	Type	Description
to_crypt	*int list*	The converted int list since an ascii string
table	*list of list*	An array containing all fully builded Base Table
base	*int*	Define the Base index

Returns **str list** : A string list containing all the base crypted values. Must be used as a crypted list.

3.11.2 Source Code

```
res = []
for i in range(0,len(to_crypt)):
        res.append(table[base][to_crypt[i]])
return res
```

3.12 local_table_dico

```
def local_table_dico(table2,base,rangeB)
```

3.12.1 Algorithm

Utils method : A method to convert a Base table to Python dictionnary

Parameters	Type	Description
table2	*list of list*	An array containing all the fully builded Base table
base	*int*	Define the Base index
rangeB	*int*	Define the max step of incrementation

Returns **Dictionnary** : A dictionnary representing the specified Base table

3.12.2 Source Code

```
str_base={}
res = {}
if(rangeB>base**2):
        rangeB=base**2
for i in range (0,rangeB):
        str_base[i]=table2[base][i]
return str_base
```

3.13 limit_range

```
def limit_range(Range,base)
```

3.13.1 Algorithm

Utils method : A method to limit the Base range

Parameters	Type	Description
Range	*int*	The range as a limit
base	*int*	The current Base index

Returns **int** : The limited by range res.

3.13.2 Source Code

```
res=0
if(Range>base**2):
        res=base**2
else:
        res=Range
return res
```

3.14 base_key

```
def base_key(int_chaine)
```

3.14.1 Algorithm

This is the key builder.

Parameters	Type	Description
int_chaine	*int list*	The base index list as a starting builder for key

Returns **int list** : the builded key from index base list.

3.14.2 Source Code

```
res=[]
for i in range (0,len(int_chaine)):
        tmp=((int_chaine[i]*int_chaine[len(int_chaine)-i-1]+10)%36)
        if(tmp<10):
                tmp+=10
        res.append(tmp)
return res
```

3.15 vec_poids

```
def vec_poids(int_chaine)
```

3.15.1 Algorithm

Compute the vectorial cumulated weight of the list.

Parameters	Type	Description
int_chaine	*int list*	The integer list to treat

Returns int list : The computed accumulated weigth integer list

3.15.2 Source Code

```
res = []
res.append(int_chaine[0])
for i in range(1,len(int_chaine)):
        res.append(res[i-1]+int_chaine[i])
return res
```

3.16 vec_1_poids

```
def vec_1_poids(vec_poids)
```

3.16.1 Algorithm

Compute the inverse of the vectorial cumulated weigth computation.

Parameters	Type	Description
vec_poids	*int list*	The weigth as an integer list

Returns int list : The computed list containing the inverse operation of vec_poids method

3.16.2 Source Code

```
res=[]
for i in range (0,len(vec_poids)):
        res.append(1/vec_poids[i])
return res
```

3.17 equa_2_nd

```
def equa_2_nd(a,b,c)
```

3.17.1 Algorithm

Utils : An 2nd order equation solver

Parameters	Type	Description
a	*int / float*	The a coefficient
b	*int / float*	The b coefficient
c	*int / float*	The c coefficient

Returns int / float : The solved equation positive root

3.17.2 Source Code

```
res = 0
racine1 = 0.0
racine2 = 0.0
delta = b**2-4*a*c
if(delta>0):
        racine1 = (-b+m.sqrt(delta))/2*a
        racine2 = (-b-m.sqrt(delta))/2*a
if(racine1>0):
        res = int(racine1)
else:
        res = int(racine2)
return res
```

3.18 multlist

```
def multlist(a,b)
```

3.18.1 Algorithm

Utils : A point by point list multiplier

Parameters	Type	Description
a	*int/float list*	The list to multiply
b	*int/float list*	The list to multiply

Returns **int / float list** : The computed point by point multiplication

3.18.2 Source Code

```
res = []
if(len(a)!=len(b)):
        return []
else:
        for i in range(0,len(a)):
                res.append(a[i]*b[i])
return res
```

3.19 transpose_base

```
def transpose_base(liste,key,table)
```

3.19.1 Algorithm

A method to transpose an integer list to the corresponding key's base index => The result will be a succession of transposed values from differents integers to differents base

Parameters	Type	Description
liste	*list*	the integer converted since ASCII list
key	*list*	The Base index list as key
table	*list*	The full Base Table recursively builded

Returns **str list** : The crypted list as String list

3.19.2 Source Code

```
res = []
if(len(liste)!=len(key)):
        return []
else :
        for i in range (0,len(liste)):
                if(key[i]==10):
                        res.append(liste[i])
                else:
                        res.append(table[key[i]-2][liste[i]])
return res
```

3.20 inv_transpose_base

```
def inv_transpose_base(liste,key,table)
```

3.20.1 Algorithm

The inverse method to decrypt a str list of base transposed values

Parameters	Type	Description
liste	*str list*	The crypted list as String list
key	*int list*	The Base index list as key
table	*int list*	The full Base table recursively builded

Returns int list : The decrypted list as integers

3.20.2 Source Code

```
res = []
if(len(liste)!=len(key)):
        return []
else:
        for i in range(0,len(liste)):
                if(key[i]==10):
                        res.append(int(liste[i]))
                else:
                        res.append(int(table[key[i]-2].index(liste[i])))
return res
```

3.21 crypt_procedure

```
def crypt_procedure(chaine,table)
```

3.21.1 Algorithm

The crypter manager to orchestrate the crypting procedure. It works from these steps:

- *We convert the given ascii string as integer list*
- *We compute the Base index list as key from the converted integer list*
- *We build the second part of the key since the mirror of the Base index list*
- *We compute the cumulated weight of the integer list*
- *We compute the point by point multiplication between cumulated weigth list and original integer list*
- *We transpose the multiplied list into the given specified Base from the key*
- *We associate the crypted strin to the key as return*

Parameters	Type	Description
chaine	*string*	The string to crypt
table	*list of list*	The Base Table recursively builded

Returns **list tuple (crypt_lst,key)** : The couple crypted string and key as result. It permits to decrypt any message.

3.21.2 Source Code

```
int_chaine = ascii_to_int(chaine)
base_keyy  = base_key(int_chaine)
if(len(base_keyy)%2==0):
        key=base_keyy[0:int(len(base_keyy)/2)]
else:
        key=base_keyy[0:int((len(base_keyy)/2)+1)]
vec_poid   = vec_poids(int_chaine)
crypt_lst  = multlist(int_chaine,vec_poid)
crypt_lst  = transpose_base(crypt_lst,base_keyy,table)
return(crypt_lst,key)
```

3.22 cyclik_ascii

```
def cyclik_ascii(current)
```

3.22.1 Algorithm

Compute a cyclik ascii separators into ponctuation signs

Parameters	Type	Description
current	*str*	The current poncuation separator

Returns **str** : The following separator from the defined 'sep' Set.

3.22.2 Source Code

```
sep=['!','"','#','$','%','&','(',')','*','+',',','-','.','/']
tmp=((sep.index(current)+1)%13)
res =sep[tmp]
return res
```

3.23 cyclik_ascii_lvl2

```
def cyclik_ascii_lvl2(current)
```

3.23.1 Algorithm

Compute a cyclik ascii separators into ponctuation signs. Get a second cyclic ascii set modulo length

Parameters	Type	Description
current	*str*	The current poncuation separator

Returns **str** : The following separator from the defined 'sep' Set.

3.23.2 Source Code

```
sep=[":",";","<","=",">","?","@"]
tmp=((sep.index(current)+1)%6)
res =sep[tmp]
return res
```

3.24 crypt_final

```
def crypt_final(tuple)
```

3.24.1 Algorithm

The layout procedure to organise crypting results.

Parameters	Type	Description
tuple	*tuple*	List couple representing the crypted strin and the associated key

Returns **str** : The crypted list as a string with correct separators

3.24.2 Source Code

```
sept=['!','"','#','$','%','&','(',')','*','+',',','-','.','/']
res = ''
sep =sept[int(int_chaine[1]*m.cos(int_chaine[0]))%13]
crypt=tuple[0]
key=tuple[1]
for i in range (0,len(crypt)):
        res+=sep+str(crypt[i])
        sep=cyclik_ascii(sep)
return res
```

3.25 crypt_final_long

```
def crypt_final_long(tuple)
```

3.25.1 Algorithm

Chaining the final-level algorithm to get complex crypto-procedure

Parameters	Type	Description
tuple	*tuple*	List couple representing the crypted string and the associated key

Returns **str** : The full second level crypted string

3.25.2 Source Code

```
sept=['!','"','#','$','%','&','(',')','*','+',',','-','.','/']
res = ''
sep =sept[int(int_chaine[1]*m.cos(int_chaine[0]))%13]
for i in range (0,len(liste)):
        res+=sep+str(liste[i])
        sep=cyclik_ascii(sep)
return res
```

3.26 slurp

```
def slurp(chaine)
```

3.26.1 Algorithm

This method allow us to rebuild a str list of crypted terms using separators set.

Parameters	Type	Description
chaine	*str*	The raw string crypted message

Returns str list : The list of crypted terms rebuilded from the raw string

3.26.2 Source Code

```
tmp=''
res = []
sep=['!','"','#','$','%','&','(',')','*','+',',','-','.','/']
for elem in chaine:
        if(not elem in sep):
                tmp+=str(elem)
        else :
                res.append(tmp)
                tmp=''
        if(elem==''):
                break
res=res[1:]
res.append(tmp)
return res
```

3.27 slurp2

```
def slurp2(chaine)
```

3.27.1 Algorithm

This method is similar of the slurp method. It defined a second level of crypting management.

Parameters	Type	Description
chaine	*str*	The raw string crypted message

Returns **str list** : The list of crypted terms rebuilded from the raw string.

3.27.2 Source Code

```
tmp=''
res = []
sep=[":",";","<","=",">","?","@"]
for elem in chaine:
        if(not elem in sep):
                tmp+=str(elem)
        else:
                res.append(tmp)
                tmp=''
        if(elem==''):
                break
res.append(tmp)
return res
```

3.28 miam

```
def miam(key)
```

3.28.1 Algorithm

Key builder from the half key as integer list. It rebuild the missing half with a mirror copy of the first one.

Parameters	Type	Description
key	*int list*	The half key as int list

Returns int list : The full key rebuilded from the half key

3.28.2 Source Code

```
tmp=''
count=1
res=[]
for this in key:
        if(count%2==0):
                tmp+=str(this)
                count=1
                res.append(tmp)
                tmp=''
        else:
                tmp=str(this)
                count+=1
for i in range(0,len(res)):
        res[i]=int(res[i])
return res
```

3.29 resolve

```
def resolve(liste)
```

3.29.1 Algorithm

This method compute the chained 2nd order equations to solve the numeric suit. It permit us to get the ASCII values as a list. To solve the system you have to instance the solver with the square root of term 0. Once theorem zero done, you will apply the equation solver with square root of the 0-term as b, a as 1 and c as -following term. The algorithm sort the roots and take only positives ones.

Parameters	Type	Description
liste	*int list*	The computed multiplied list to solve

Returns int list : A list containing solved terms.

3.29.2 Source Code

```
res = []
x = 0
tmp2 = 0
res.append(int(m.sqrt(liste[0])))
tmp=res[0]
for i in range (1,len(liste)):
        tmp2 = equa_2_nd(1,-tmp,-liste[i])
        x=tmp2-tmp
        res.append(int(x))
        tmp=tmp2
return res
```

3.30 decrypt_procedure

```
def decrypt_procedure(chaine,key,table)
```

3.30.1 Algorithm

This method manage the decrypting procedure. It is ruled by the following steps :

- **Build the full key since the key argument**
- **Split the string** since separators via slurp method
- **Apply the inv_tranpose_base method** to get the uncrypted terms
- **Solve the cumulated multiplied weigth** with the equation solver
- **Convert the int list** as result to ASCII chain

Parameters	Type	Description
chaine	*str*	The raw crypted text as string
key	*int list*	The half key as int list
table	*list of list*	The Base Table array

Returns **str** : The uncrypted text.

3.30.2 Source Code

```
res = ''
base=key[:]
tmp = []
key.reverse()
tmp = key[:]
to_find = []
to_find=slurp(chaine)
if(len(to_find)%2==0):
        base+=tmp[0:len(key)]
else:
        base+=tmp[1:len(key)]
tmp_liste=inv_transpose_base(to_find,base,table)
int_liste=resolve(tmp_liste)
res = int_to_ascii(int_liste)
return res
```

3.31 split

```
def split(chaine,seuil)
```

3.31.1 Algorithm

Split the given string argument 'chaine' into slices from threshold size 'seuil'. Each of this slices are allowed into the cryptographic algorithm.

Parameters	Type	Description
chaine	*str*	The full string to treat
seuil	*int*	Define the threshold size of the slices

Returns str list : The slices list as result

3.31.2 Source Code

```
res = []
tmp = ''
index = 0
div=int(len(chaine)/seuil)
for i in range(0,div):
        tmp=''
        for j in range(index,(index+seuil)):
                tmp+=chaine[j]
                if(j==(index+seuil-1)):
                        index=j+1
        res.append(tmp)
if((index-1)<len(chaine)):
        tmp=chaine[index:]
        res.append(tmp)
return res
```

3.32 tilps

```
def tilps(chaine)
```

3.32.1 Algorithm

The reverse method of the split function. From a given str list, we rebuild the full length string

Parameters	Type	Description
chaine	*str list*	The String slices as a list

Returns **str** : The full striing rebuilded from the slices list

3.32.2 Source Code

```
res = ''
for i in range (0,len(chaine)):
        res+=chaine[i]
return res
```

CHAPTER

FOUR

RAPTOR CRYPTOGRAPHIC ALGORITHM V3

4.1 Description of Crypter

Welcom to Raptor cryptographic help

This following instructions give you the full light on the given cryptographic algorithm "Raptor". In a firts time I will explain the main algorithm rules. Each of the function used can be found on the full source code and have a dedicated help section.

Description of the Main Raptor's Cryptographic Algorithm

4.1.1 Algorithm

This is the main algorithm of the program. It allows from a system argv string to crypt it and get a string,key couple as result. We will use this following variables to make it work :

- **table2** : A list of list containing all the necessary Base table from Base 11 to Base 37
- **Basemin** : 2 as default, it means the minimum base index to generate
- **Basemax** : 37 as default, it means the maximum base index to generate
- **chaine** : The string chain to crypt as system argv argument
- **choice** : A choice variable to manage the main loop (continue or quit)
- **Range** : Define the range of values generate into the corresponding Numeric Base a the begining

The return of the algorithm is ruled by the following variables:

- **testkey** : The final half key as key
- **raw_txt** : The final crypted strin as string.

This is the main Raptor Cryptographic Algorithm v3. It is ruled by the following steps :

- **Initialization** of differents variables and of the Base table via the table generator methods
- **Splitting** part of the given raw string as input. This string will be splitted into differents slices, wiche be crypted one by one and associated to his key via the third level separators wich define the third level of the crypting tree.
- **Crypting procedure** for each of the slices obtained by the split method above. These crypted results will be stored as a list of list, respectively a list of slices, defined by a list of crypted terms.
- **Manage the results** via slurp2 and slurp3 methods. The results are properly stored at this time to be correctly interpreted later.
- **Give a wrong path** for decrypting using some fake values to both of crypted txt and key as strings. It means any Brute force attack will be ignored.
- **Returns** the couple (crypt txt, key) wich is efficient to be decrypted by the solver.

This algorithm is stable in his domain and must be used on it. Please not to try bigger data slice and automate it via shell script if necessary. It should be used as a data crypter using a top level slicer and manager (from the shell script as exemple).

See source below to more explanation.

4.1.2 Source Code

```
import sys
import math as m
import random as r

represent=''
table2 = []
dic = {}
main_dic={}
choice = ' '
chaine=''

if(len(sys.argv)!=4):
        Basemin = 2
        Basemax = 37
        Range   = 36**2
else :
        Basemin = int(sys.argv[1])
        Basemax = int(sys.argv[2])
        Range   = int(sys.argv[3])

if(Basemin<2 or Basemax>37):
        print("Affichage impossible veuillez selectionner une plage de valeure contenue dans [2,36]")
        exit(0)

maxi=Basemax-Basemin

for i in range(Basemin,Basemax):
        table2.append(table(i,0,Range,1))

for i in range (0,len(table2)):
        table2[i]=splitTable(table2[i])

for j in range (0,len(table2)):
        table2[j]=rec_table_construct_lvl1(table2[j],j+2,1,0)
        for k in range(0,j+2):
                table2[j][k]=(str(0)+table2[j][k])
table2=rec_manage(table2)

long_chaine = []
long_crypt  = []
longi=0
seuil = 20
seuil_lvl2=70
choice = ''
userchoice=1
sep=['!','"','#','$','%','&','(',')','*','+',',','-','.','/']
sep_lvl2=[":",";","<","=",">","?","@"]
sep_lvl3=['A','B','C','D','E','F','G','H','I','J','K','L']
mesquin=['M','N','O','P','Q','R','S','T','U','V','W','X','Y','Z']
```

(continues on next page)

(continued from previous page)

```
long_long_chaine = []
tmp_long_chaine  = []
long_chaine    = []
long_crypt     = []
testc          = []
testk          = []
int_chaine     = []
lvl2_key_miam = []
tmp_crypt      = []


while(choice!='q'):
        # init_all()
        current_sep_lvl3 =  "A"
        current_sep_lvl2 =  ":"
        long_chaine   = []
        long_crypt     = []
        long_long_crypt  = []
        testc          = []
        testk          = []
        int_chaine     = []
        lvl2_key_miam = []
        long_long_chaine = []
        tmp_long_chaine  = []
        tmp_crypt        = ()
        testkey=''
        raw_txt=''
        clean_txt = ''
        longi = 0
        longii= 0

        res = ()
        if(userchoice):
                chaine = ""
                chaine=input("Veuillez entrer la chaine à crypter  (>20): ")
        if(len(chaine)>=seuil and len(chaine)<seuil_lvl2):
                long_chaine = split(chaine,seuil)
                longi+=1
        else:
                if(len(chaine)>=seuil_lvl2):
                        tmp_long_chaine = split(chaine,seuil_lvl2)
                        for i in range(0,len(tmp_long_chaine)):
                                long_long_chaine.append(split(tmp_long_chaine[i],seuil))
                        longii+=1

        if(not longi and not longii):
                res=crypt_procedure(chaine,table2)
        else :
                if(longi):
                        for i in range(0,len(long_chaine)):
                                long_crypt.append(crypt_procedure(long_chaine[i],table2))
                if(longii):
```

(continues on next page)

(continued from previous page)

```
                                for i in range (0,len(long_long_chaine)):
                                        for j in range(0,len(long_long_chaine[i])):
                                                tmp_crypt = crypt_procedure(long_long_
↪chaine[i][j],table2)
                                                long_long_crypt.append(tmp_crypt)
                if(not longi and not longii):
                        testc = res[0]
                        testk = res[1]
                else :
                        if (longi):
                                for i in range (0,len(long_crypt)):
                                        for j in range(0,len(long_crypt[i][0])):
                                                testc.append(str(long_crypt[i][0][j]))
                                        for k in range(0,len(long_crypt[i][1])):
                                                testk.append(str(long_crypt[i][1][k]))
                                        current_sep_lvl2=cyclik_ascii_lvl2(current_sep_lvl2)
                                        testc[-1]+=current_sep_lvl2
                                        testk[-1]+=current_sep_lvl2
                        if(longii):

                                for l in range (0,len(long_long_crypt)):
                                        for j in range(0,len(long_long_crypt[l][0])):
                                                testc.append(str(long_long_crypt[l][0][j]))
                                        for k in range(0,len(long_long_crypt[l][1])):
                                                testk.append(str(long_long_crypt[l][1][k]))
                                        current_sep_lvl2=cyclik_ascii_lvl2(current_sep_lvl2)
                                        testc[-1]+=current_sep_lvl2
                                        testk[-1]+=current_sep_lvl2
                                        if(len(long_long_crypt[l][0])<seuil):
                                                current_sep_lvl3=cyclik_ascii_lvl3(current_sep_
↪lvl3)

                                                testc[-1]+=current_sep_lvl3
                                                testk[-1]+=current_sep_lvl3
                int_chaine=(ascii_to_int(chaine))
                for i in range(0,len(testk)):
                        testkey+=str(testk[i])

                if(not longi and not longii):
                        raw_txt = crypt_final(res,int_chaine)
                else:
                        raw_txt += crypt_final_long(testc,int_chaine)
                raw_txt=mesqui(raw_txt,seuil)
                testkey=mesqui(testkey,seuil)
                print("Chaine cryptée : \n")
                print(raw_txt)
                print("Clé unique : \n")
                print(testkey)
                choice=input("c)ontinuer ou q)uitter")
                if(choice!='q'):
                        userchoice+=1
```

4.2 Description of De-crypter

Description of the Main Raptor's Cryptographic Algorithm

4.2.1 Algorithm

This is the main solver algorithm program. It allow us to decrypt datas slices crypted with the version 1 of the Raptor Cryptographic Algorithm. To solve I need thse following variables :

- **raw_txt** : The input crypted string storage
- **Basemin** : The minimum Base index
- **Basemax** : The maximum Base index
- **table2** : The list of list containing the Base Table
- **testkey** : The key of the algorithm, the decrypting process absolutely need this key.

The solving procedure is ruled by the following steps:

- **Generating the Base Table** and store it into my table2 variable
- **Getting inputs** known as crypted string and his associated key.
- **Organize data slice** removing separators via the slurps methods
- **Decrypting process** using the decrypt_procedure method (see documentation)
- **Store and return** the results of decrypting process

4.2.2 Source Code

```
import sys
import math as m
import random as r

represent=''
table2 = []
dic = {}
main_dic={}
choice = ' '
chaine=''

if(len(sys.argv)!=4):
        Basemin = 2
        Basemax = 37
        Range   = 36**2
else :
        Basemin = int(sys.argv[1])
        Basemax = int(sys.argv[2])
        Range   = int(sys.argv[3])
```

(continues on next page)

(continued from previous page)

```
if(Basemin<2 or Basemax>37):
        print("Affichage impossible veuillez selectionner une plage de valeure contenue␣
↪dans [2,36]")
        exit(0)

maxi=Basemax-Basemin

for i in range(Basemin,Basemax):
        table2.append(table(i,0,Range,1))

for i in range (0,len(table2)):
        table2[i]=splitTable(table2[i])

for j in range (0,len(table2)):
        table2[j]=rec_table_construct_lvl1(table2[j],j+2,1,0)
        for k in range(0,j+2):
                table2[j][k]=(str(0)+table2[j][k])
table2=rec_manage(table2)

long_chaine = []
long_crypt  = []
longi=0
seuil = 20
seuil_lvl2=70
choice = ''
userchoice=0
sep=['!','"','#','$','%','&','(',')','*','+',',','-','.','/']
sep_lvl2=[":",";","<","=",">","?","@"]
sep_lvl3=['A','B','C','D','E','F','G','H','I','J','K','L']
mesquin=['M','N','O','P','Q','R','S','T','U','V','W','X','Y','Z']

long_long_chaine = []
tmp_long_chaine  = []
long_chaine    = []
long_crypt     = []
testc          = []
testk          = []
int_chaine     = []
lvl2_key_miam = []
tmp_crypt      = []


while(choice!='q'):
        # init_all()
        current_sep_lvl3 =  "A"
        current_sep_lvl2 =  ":"
        long_chaine[:]    = []
        long_crypt[:]     = []
        long_long_crypt  = []
        testc[:]          = []
        testk[:]          = []
```

(continues on next page)

(continued from previous page)

```
        int_chaine[:]    = []
        lvl2_key_miam[:] = []
        long_long_chaine[:] = []
        tmp_long_chaine[:]  = []
        tmp_crypt        = ()
        testkey=''
        raw_txt=''
        clean_txt = ''
        longi = 0
        longii= 0

        res = ()
        raw_txt=input("Chaine cryptée : \n")
        testkey=input("Clé unique : \n")
        if(len(raw_txt)/5>=seuil and len(raw_txt)/5<seuil_lvl2):
                longi+=1
        if(len(raw_txt)/5>=seuil_lvl2):
                longii+=1
        raw_txt = slurp3(raw_txt)
        testkey = slurp3(testkey)
        if(not longi and not longii):
                testkey=miam(testkey)
                clean_txt = decrypt_procedure(raw_txt,testkey,table2)
        else:
                if(longi):
                        lvl2_liste = []
                        lvl2_key   = []
                        lvl2_liste = slurp2(raw_txt)
                        lvl2_key   = slurp2(testkey)
                        lvl2_key_miam = []
                        for i in range (0,len(lvl2_key)):
                                lvl2_key_miam.append(miam(lvl2_key[i]))
                        for i in range (0,len(lvl2_liste)-1):
                                clean_txt+= decrypt_procedure(lvl2_liste[i],lvl2_key_
↪miam[i],table2)

                if(longii):
                        lvl3_liste = []
                        lvl3_key   = []
                        lvl3_liste = slurp4(raw_txt)
                        lvl3_key   = slurp4(testkey)
                        lvl2_liste = []
                        lvl2_key   = []
                        lvl2_key_miam = []
                        final_key  = []
                        for i in range (0,len(lvl3_key)):
                                lvl2_key.append(slurp2(lvl3_key[i]))
                        for i in range (0,len(lvl3_liste)-1):
                                lvl2_liste.append(slurp2(lvl3_liste[i]))
                        for i in range(0,len(lvl2_key)-1):
                                lvl2_key_miam[:] = []
                                for j in range (0,len(lvl2_key[i])):
```

(continues on next page)

(continued from previous page)

```
                                                        lvl2_key_miam.append(miam(lvl2_key[i][j]))
                                                del lvl2_key_miam[-1]
                                                final_key.append(lvl2_key_miam)
                                                for k in range (0,len(lvl2_liste[i])-1):
                                                        clean_txt+=decrypt_procedure(lvl2_liste[i][k],
↪final_key[0][k],table2)

        print("Chaine décryptée : \n")
        print(clean_txt)
        choice=input("c)ontinuer ou q)uitter")
        if(choice!='q'):
                userchoice+=1
```

4.3 Reverse

```
def reverse(s)
```

4.3.1 Algorithm

A function to reverse a string as argument.

Parameter	Type	Description
s	*String*	The string to reverse

Returns **str** : The reversed string

4.3.2 Source Code

```
str= ""
for i in s:
        str=i+str
return str
```

4.4 splitTable

```
def splitTable(table)
```

4.4.1 Algorithm

Split a string as array from the given separator.

Parameters	Type	Description
table	*string*	The list to split

Returns **list** : The splitted list

4.4.2 Source Code

```
local_list=table.split('\n')
res_list=[]
for i in range (0,len(local_list)):
        res_list.append(local_list[i])
return res_list
```

4.5 table

```
def table(base,debut,fin,inc)
```

4.5.1 Algorithm

Base table recursive builder. The generated Base table array is defined via :

- **base** : Define the base to begin the table
- **debut** : Define the first value of Base table
- **fin** : Define the last value of Base table
- **inc** : Define the incrementation step

Parameters	Type	Description
base	*int*	The first base of the table
debut	*int*	The first value of the table in the given base
fin	*int*	The last value of the table in the given base
inc	*int*	The value of incrementation step

Returns Str : A string containing all the base generated representing the array (see conversion later)

4.5.2 Source Code

```
represent=''
letter='a'
powIndex=0
count=0
if(fin>10*base):
        fin=10*base
for i in range(debut,fin):
        current=i
        if(i<base):
                if(i<10):
                        represent+=str(i)
                else:
                        represent+=letter
                        letter=chr(ord(letter)+1)
                if(i==base-1):
                        letter='a'
        else:
                tmp=''
                while(current/base!=0):
                        count=powIndex*10*base
                        if(not current%(10*base)):
                                                powIndex+=1
```

(continues on next page)

(continued from previous page)

```
                        if(base<10):
                                tmp+=str(current%base)
                        else:
                                if(current%base<10):
                                        tmp+=str(current%base)
                                else:
                                        tmp+=letter
                                        if(count==0):
                                                letter=chr(ord(letter)+1)
                                        else:
                                                count-=1
                                        if(current%base==base-1):
                                                letter='a'
                        current=int(current/base)
                represent+=reverse(tmp) #comment this lonely line to run out the program␣
↪:/
        represent+="\n"
return represent
```

4.6 rec_table_construct_lvl1

```
def rec_table_construct_lvl1(table,base,powindex,last)
```

4.6.1 Algorithm

Recursive Construction method from the Base table. The recursive algorithm permit to edit much larger array from existing original base table. Ths algorithm must be used as the init loop of the final recursive method (see rec_manage method)

Parameters	Type	Description
table	*list*	The Base table array
base	*int*	The current numeric base as integer
powindex	*int*	The pow index as integer
last	*int*	unused

Returns **list** : The Recursively builded Base table as list

4.6.2 Source Code

```
lettrebase=table[10:base]
if(powindex == 1):
        del table[10*base]
res=table[:]
for i in range (len(table)-1,base**2-1):
        if(i%base==(base-1) and i!=len(table)-1):
                powindex+=1
        res.append(lettrebase[powindex-1]+str(table[(i-len(table)+1)%base]))
return res
```

4.7 rec_table_construct_final

```
def rec_table_construct_final(table,base,lvl)
```

4.7.1 Algorithm

Recursive Construction method from the Base table. The recursive algorithm manage array building since 2 levels of recursive construction. => Do not use for the first recursive building loop

Parameters	Type	Description
table	*list*	The first recursive level builded Base table
base	*int*	The base to treat as integer
lvl	*int*	The level of recursivity in construction

Returns list : The fully specified level recursivity builded Base table

4.7.2 Source Code

```
res=[]
basetable=table[0:base]
for i in range(0,len(basetable)):
        basetable[i]=basetable[i][lvl:]
for eat in basetable:
        for this in table:
                res.append(eat+this)
return res
```

4.8 rec_manage

```
def rec_manage(table)
```

4.8.1 Algorithm

A recursivity manager to build properly the base table. It must be used to map the numeric values into base values. This method allow contruction of hundreds of thousand values table

Parameters	Type	Description
table	*list*	The initial Base table to complete

Returns list : The fully builded Base table

4.8.2 Source Code

```
j=0
for i in range(9,len(table)):
        table2=table[i][:]
        table2=rec_table_construct_lvl1(table2,i+2,1,0)
        table2=rec_table_construct_final(table2,i+2,1)
        table2=rec_table_construct_final(table2,i+2,2)
        table[i]=table2[:]
for i in range (9,18):
        j=3
        table2=table[i][:]
        while(len(table2)<1000000):
                table2=rec_table_construct_final(table2,i+2,j)
                j+=1
        table[i]=table2[:]
return table
```

4.9 ascii_to_int

```
def ascii_to_int(chaine)
```

4.9.1 Algorithm

Utils method : ascii to integer converter.

Parameters	Type	Description
chaine	*str*	The string to convert

Returns **list** : A list containing all integers values since ASCII.

4.9.2 Source Code

```
res = []
for letter in chaine:
        res.append(ord(letter))
return res
```

4.10 int_to_ascii

```
def int_to_ascii(crypt)
```

4.10.1 Algorithm

Utils method : integer to ascii converter.

Description	Type	Description
crypt	*int list*	The int list to convert

Returns str : The converted ASCII string since int list.

4.10.2 Source Code

```
res = ''
for i in range (0,len(crypt)):
        res+=chr(crypt[i])
return res
```

4.11 cryptChaine

```
def cryptChaine(to_crypt,table,base)
```

4.11.1 Algorithm

The simple method to crypt an ascii string as integer list.

Parameters	Type	Description
to_crypt	*int list*	The converted int list since an ascii string
table	*list of list*	An array containing all fully builded Base Table
base	*int*	Define the Base index

Returns str list : A string list containing all the base crypted values. Must be used as a crypted list.

4.11.2 Source Code

```
res = []
for i in range(0,len(to_crypt)):
        res.append(table[base][to_crypt[i]])
return res
```

4.12 local_table_dico

```
def local_table_dico(table2,base,rangeB)
```

4.12.1 Algorithm

Utils method : A method to convert a Base table to Python dictionnary

Parameters	Type	Description
table2	*list of list*	An array containing all the fully builded Base table
base	*int*	Define the Base index
rangeB	*int*	Define the max step of incrementation

Returns Dictionnary : A dictionnary representing the specified Base table

4.12.2 Source Code

```
str_base={}
res = {}
if(rangeB>base**2):
        rangeB=base**2
for i in range (0,rangeB):
        str_base[i]=table2[base][i]
return str_base
```

4.13 limit_range

```
def limit_range(Range,base)
```

4.13.1 Algorithm

Utils method : A method to limit the Base range

Parameters	Type	Description
Range	*int*	The range as a limit
base	*int*	The current Base index

Returns int : The limited by range res.

4.13.2 Source Code

```
res=0
if(Range>base**2):
        res=base**2
else:
        res=Range
return res
```

4.14 base_key

```
def base_key(int_chaine)
```

4.14.1 Algorithm

This is the key builder.

Parameters	Type	Description
int_chaine	*int list*	The base index list as a starting builder for key

Returns **int list** : the builded key from index base list.

4.14.2 Source Code

```
res=[]
for i in range (0,len(int_chaine)):
        tmp=((int_chaine[i]*int_chaine[len(int_chaine)-i-1]+10)%36)
        if(tmp<10):
                tmp+=10
        res.append(tmp)
return res
```

4.15 vec_poids

```
def vec_poids(int_chaine)
```

4.15.1 Algorithm

Compute the vectorial cumulated weight of the list.

Parameters	Type	Description
int_chaine	*int list*	The integer list to treat

Returns int list : The computed accumulated weigth integer list

4.15.2 Source Code

```
res = []
res.append(int_chaine[0])
for i in range(1,len(int_chaine)):
        res.append(res[i-1]+int_chaine[i])
return res
```

4.16 vec_1_poids

```
def vec_1_poids(vec_poids)
```

4.16.1 Algorithm

Compute the inverse of the vectorial cumulated weigth computation.

Parameters	Type	Description
vec_poids	*int list*	The weigth as an integer list

Returns int list : The computed list containing the inverse operation of vec_poids method

---

4.16.2 Source Code

```
res=[]
for i in range (0,len(vec_poids)):
        res.append(1/vec_poids[i])
return res
```

4.17 equa_2_nd

```
def equa_2_nd(a,b,c)
```

4.17.1 Algorithm

Utils : An 2nd order equation solver

Parameters	Type	Description
a	*int / float*	The a coefficient
b	*int / float*	The b coefficient
c	*int / float*	The c coefficient

Returns int / float : The solved equation positive root

4.17.2 Source Code

```
res = 0
racine1 = 0.0
racine2 = 0.0
delta = b**2-4*a*c
if(delta>0):
        racine1 = (-b+m.sqrt(delta))/2*a
        racine2 = (-b-m.sqrt(delta))/2*a
if(racine1>0):
        res = int(racine1)
else:
        res = int(racine2)
return res
```

4.18 multlist

```
def multlist(a,b)
```

4.18.1 Algorithm

Utils : A point by point list multiplier

Parameters	Type	Description
a	*int/float list*	The list to multiply
b	*int/float list*	The list to multiply

Returns **int / float list** : The computed point by point multiplication

4.18.2 Source Code

```
res = []
if(len(a)!=len(b)):
        return []
else:
        for i in range(0,len(a)):
                res.append(a[i]*b[i])
return res
```

4.19 transpose_base

```
def transpose_base(liste,key,table)
```

4.19.1 Algorithm

A method to transpose an integer list to the corresponding key's base index => The result will be a succession of transposed values from differents integers to differents base

Parameters	Type	Description
liste	*list*	the integer converted since ASCII list
key	*list*	The Base index list as key
table	*list*	The full Base Table recursively builded

Returns str list: The crypted list as String list

4.19.2 Source Code

```
res = []
if(len(liste)!=len(key)):
        return []
else :
        for i in range (0,len(liste)):
                if(key[i]==10):
                        res.append(liste[i])
                else:
                        res.append(table[key[i]-2][liste[i]])
return res
```

4.20 inv_transpose_base

```
def inv_transpose_base(liste,key,table)
```

4.20.1 Algorithm

The inverse method to decrypt a str list of base transposed values

Parameters	Type	Description
liste	*str list*	The crypted list as String list
key	*int list*	The Base index list as key
table	*int list*	The full Base table recursively builded

Returns **int list** : The decrypted list as integers

4.20.2 Source Code

```
res = []
if(len(liste)!=len(key)):
        return []
else:
        for i in range(0,len(liste)):
                if(key[i]==10):
                        res.append(int(liste[i]))
                else:
                        res.append(int(table[key[i]-2].index(liste[i])))
return res
```

4.21 crypt_procedure

```
def crypt_procedure(chaine,table)
```

4.21.1 Algorithm

The crypter manager to orchestrate the crypting procedure. It works from these steps:

- *We convert the given ascii string as integer list*
- *We compute the Base index list as key from the converted integer list*
- *We build the second part of the key since the mirror of the Base index list*
- *We compute the cumulated weight of the integer list*
- *We compute the point by point multiplication between cumulated weigth list and original integer list*
- *We transpose the multiplied list into the given specified Base from the key*
- *We associate the crypted strin to the key as return*

Parameters	Type	Description
chaine	*string*	The string to crypt
table	*list of list*	The Base Table recursively builded

Returns list tuple : The couple crypted string and key as result. It permits to decrypt any message.

4.21.2 Source Code

```
int_chaine = ascii_to_int(chaine)
base_keyy  = base_key(int_chaine)
if(len(base_keyy)%2==0):
        key=base_keyy[0:int(len(base_keyy)/2)]
else:
        key=base_keyy[0:int((len(base_keyy)/2)+1)]
vec_poid   = vec_poids(int_chaine)
crypt_lst  = multlist(int_chaine,vec_poid)
crypt_lst  = transpose_base(crypt_lst,base_keyy,table)
return(crypt_lst,key)
```

4.22 cyclik_ascii

```
def cyclik_ascii(current)
```

4.22.1 Algorithm

Compute a cyclik ascii separators into ponctuation signs

Parameters	Type	Description
current	*str*	The current poncuation separator

Returns str : The following separator from the defined 'sep' Set.

4.22.2 Source Code

```
sep=['!','"','#','$','%','&','(',')','*','+',',','-','.','/']
tmp=((sep.index(current)+1)%13)
res =sep[tmp]
return res
```

4.23 cyclik_ascii_lvl2

```
def cyclik_ascii_lvl2(current)
```

4.23.1 Algorithm

Compute a cyclik ascii separators into ponctuation signs. Get a second cyclic ascii set modulo length

Parameters	Type	Description
current	*str*	The current poncuation separator

Returns **str** : The following separator from the defined 'sep' Set.

4.23.2 Source Code

```
sep=[":",";","<","=",">","?","@"]
tmp=((sep.index(current)+1)%6)
res =sep[tmp]
return res
```

4.24 cyclik_ascii_lvl3

```
def cyclik_ascii_lvl3(current)
```

4.24.1 Algorithm

Compute a cyclik ascii separators into Upper letters from A to L. Get a third cyclic ascii set modulo length

Parameters	Type	Description
current	*str*	The current poncuation separator

Returns str : The following separator from the defined 'sep' Set.

4.24.2 Source Code

```
sep=['A','B','C','D','E','F','G','H','I','J','K','L']
tmp=r.randint(0,11)
res = sep[tmp]
return res
```

4.25 cyclik_ascii_mesquin

```
def cyclik_ascii_mesquin(current,int_chaine)
```

4.25.1 Algorithm

Compute a cyclik ascii separators into Upper letters from M to Z. Get a third cyclic ascii set modulo length

Parameters	Type	Description
current	*str*	The current poncuation separator

Returns str : The following separator from the defined 'sep' Set.

4.25.2 Source Code

```
mesquin=['M','N','O','P','Q','R','S','T','U','V','W','X','Y','Z']
tmp=r.randint(0,11)
res=mesquin[tmp]
return res
```

4.26 crypt_final

```
def crypt_final(tuple,int_chaine)
```

4.26.1 Algorithm

The layout procedure to organise crypting results.

Parameters	Type	Description
tuple	*tuple*	List couple representing the crypted strin and the associated key

Returns **str** : The crypted list as a string with correct separators

4.26.2 Source Code

```
sept=['!','"','#','$','%','&','(',')','*','+',',','-','.','/']
res = ''
sep =sept[int(int_chaine[1]*m.cos(int_chaine[0]))%13]
crypt=tuple[0]
key=tuple[1]
for i in range (0,len(crypt)):
        res+=sep+str(crypt[i])
        sep=cyclik_ascii(sep)
return res
```

4.27 crypt_final_long

```
def crypt_final_long(liste,int_chaine)
```

4.27.1 Algorithm

Chaining the final-level algorithm to get complex crypto-procedure

Parameters	Type	Description
tuple	*tuple*	List couple representing the crypted string and the associated key

Returns **str** : The full second level crypted string

4.27.2 Source Code

```
sept=['!','"','#','$','%','&','(',')','*','+',',','-','.','/']
res = ''
sep =sept[int(int_chaine[1]*m.cos(int_chaine[0]))%13]
for i in range (0,len(liste)):
        res+=sep+str(liste[i])
        sep=cyclik_ascii(sep)
return res
```

4.28 slurp

```
def slurp(chaine)
```

4.28.1 Algorithm

This method allow us to rebuild a str list of crypted terms using separators set.

Parameters	Type	Description
chaine	*str*	The raw string crypted message

Returns str list : The list of crypted terms rebuilded from the raw string

4.28.2 Source Code

```
tmp=''
res = []
sep=['!','"','#','$','%','&','(',')','*','+',',','-','.','/']
for elem in chaine:
        if(not elem in sep):
                tmp+=str(elem)
        else :
                res.append(tmp)
                tmp=''
        if(elem==''):
                break
res=res[1:]
res.append(tmp)
return res
```

4.29 slurp2

```
def slurp2(chaine)
```

4.29.1 Algorithm

This method is similar of the slurp method. It defined a second level of crypting management.

Parameters	Type	Description
chaine	*str*	The raw string crypted message

Returns str list: The list of crypted terms rebuilded from the raw string.

4.29.2 Source Code

```
tmp=''
res = []
sep=[":",";","<","=",">","?","@"]
for elem in chaine:
        if(not elem in sep):
                tmp+=str(elem)
        else:
                res.append(tmp)
                tmp=''
        if(elem==''):
                break
res.append(tmp)
return res
```

4.30 slurp3

```
def slurp3(chaine)
```

4.30.1 Algorithm

This method is similar of the slurp2 method. It defined a third level of crypting management.

Parameters	Type	Description
chaine	*str*	The raw string crypted message

Returns **str list** : The list of crypted terms rebuilded from the raw string.

4.30.2 Source Code

```
tmp=''
mesquin=['M','N','O','P','Q','R','S','T','U','V','W','X','Y','Z']
for elem in chaine:
        if(not elem in mesquin):
                tmp+=str(elem)
return tmp
```

4.31 slurp4

```
def slurp4(chaine)
```

4.31.1 Algorithm

This method is similar of the slurp2 method. It defined a third level of crypting management.

Parameters	Type	Description
chaine	*str*	The raw string crypted message

Returns **str list** : The list of crypted terms rebuilded from the raw string.

4.31.2 Source Code

```
tmp=''
res = []
sep=['A','B','C','D','E','F','G','H','I','J','K','L']
for elem in chaine:
        if(not elem in sep):
                tmp+=str(elem)
        else:
                res.append(tmp)
                tmp=''
        if(elem==''):
                break
res.append(tmp)
return res
```

4.32 miam

```
def miam(key)
```

4.32.1 Algorithm

Key builder from the half key as integer list. It rebuild the missing half with a mirror copy of the first one.

Parameters	Type	Description
key	*int list*	The half key as int list

Returns int list : The full key rebuilded from the half key

4.32.2 Source Code

```
tmp=''
count=1
res=[]
for this in key:
        if(count%2==0):
                tmp+=str(this)
                count=1
                res.append(tmp)
                tmp=''
        else:
                tmp=str(this)
                count+=1
for i in range(0,len(res)):
        res[i]=int(res[i])
return res
```

4.33 resolve

```
def resolve(liste)
```

4.33.1 Algorithm

This method compute the chained 2nd order equations to solve the numeric suit. It permit us to get the ASCII values as a list. To solve the system you have to instance the solver with the square root of term 0. Once theorem zero done, you will apply the equation solver with square root of the 0-term as b, a as 1 and c as -following term. The algorithm sort the roots and take only positives ones.

Parameters	Type	Description
liste	*int list*	The computed multiplied list to solve

Returns **int list** : A list containing solved terms.

4.33.2 Source Code

```
res = []
x = 0
tmp2 = 0
res.append(int(m.sqrt(liste[0])))
tmp=res[0]
for i in range (1,len(liste)):
        tmp2 = equa_2_nd(1,-tmp,-liste[i])
        x=tmp2-tmp
        res.append(int(x))
        tmp=tmp2
return res
```

4.34 decrypt_procedure

```
def decrypt_procedure(chaine,key,table)
```

4.34.1 Algorithm

This method manage the decrypting procedure. It is ruled by the following steps :

- **Build the full key** since the key argument
- **Split the string** since separators via slurp method
- **Apply the inv_tranpose_base method** to get the uncrypted terms
- **Solve the cumulated multiplied weigth** with the equation solver
- **Convert the int list** as result to ASCII chain

Parameters	Type	Description
chaine	*str*	The raw crypted text as string
key	*int list*	The half key as int list
table	*list of list*	The Base Table array

Returns str : The uncrypted text.

4.34.2 Source Code

```
res = ''
base=key[:]
tmp = []
key.reverse()
tmp = key[:]
to_find = []
to_find=slurp(chaine)
if(len(to_find)%2==0):
        base+=tmp[0:len(key)]
else:
        base+=tmp[1:len(key)]
# Complexify
tmp_liste=inv_transpose_base(to_find,base,table)
int_liste=resolve(tmp_liste)
res = int_to_ascii(int_liste)
return res
```

4.35 split

```
def split(chaine,seuil)
```

4.35.1 Algorithm

Split the given string argument 'chaine' into slices from threshold size 'seuil'. Each of this slices are allowed into the cryptographic algorithm.

Parameters	Type	Description
chaine	*str*	The full string to treat
seuil	*int*	Define the threshold size of the slices

Returns **str list** : The slices list as result

4.35.2 Source Code

```
res = []
tmp = ''
index = 0
div=int(len(chaine)/seuil)
for i in range(0,div):
        tmp=''
        for j in range(index,(index+seuil)):
                tmp+=chaine[j]
                if(j==(index+seuil-1)):
                        index=j+1
        res.append(tmp)
if((index-1)<len(chaine)):
        tmp=chaine[index:]
        res.append(tmp)
return res
```

4.36 tilps

```
def tilps(chaine)
```

4.36.1 Algorithm

The reverse method of the split function. From a given str list, we rebuild the full length string

Parameters	Type	Description
chaine	*str list*	The String slices as a list

Returns str : The full striing rebuilded from the slices list

---

4.36.2 Source Code

```
res = ''
for i in range (0,len(chaine)):
        res+=chaine[i]
return res
```

4.37 mesqui

```
def mesqui(txt,seuil)
```

4.37.1 Algorithm

This method is used to create a wrong path of decrypting method. Using a similar Separators terms, I define a 'fake' terms list wich have absolutely no meanings for the rest of the algorithm. Using it as the last step of algorithm, it doesn't allow any brute force attack to decrypt. The threshold value 'seuil' will define the amount of distribution of fake separators.

Parameters	Type	Description
txt	*str*	The raw string to treat
seuil	*int*	The threshold variable to assign the 'fake terms' length

Returns **str** : The fully 'fake splitted' crypted string

4.37.2 Source Code

```
mesquin=['M','N','O','P','Q','R','S','T','U','V','W','X','Y','Z']
res=''
sep='M'
for i in range(0,len(txt)):
        res+=txt[i]
        if(i%int((seuil))==0):
                res+=sep
                sep=cyclik_ascii_mesquin(sep,int_chaine)
return res
```

CHAPTER

FIVE

RAPTOR CRYPTOGRAPHIC ALGORITHM V3.1

This is the main Raptor Cryptographic Algorithm v3.1. It use the base_opt module to build Base Table array and follow the same principe of olders ones adding the new feature of dynamically complement results values.

5.1 Description of De-crypter

Welcom to Raptor cryptographic help

This following instructions give you the full light on the given cryptographic algorithm "Raptor". In a firts time I will explain the main algorithm rules. Each of the function used can be found on the full source code and have a dedicated help section.

Description of the Main Raptor's Cryptographic Algorithm

5.1.1 Algorithm

This is the main algorithm of the program. It allows from a system argv string to crypt it and get a string,key couple as result. We will use this following variables to make it work :

- **table2** : A list of list containing all the necessary Base table from Base 11 to Base 37
- **Basemin** : 2 as default, it means the minimum base index to generate
- **Basemax** : 37 as default, it means the maximum base index to generate
- **chaine** : The string chain to crypt as system argv argument
- **choice** : A choice variable to manage the main loop (continue or quit)
- **Range** : Define the range of values generate into the corresponding Numeric Base a the begining

The return of the algorithm is ruled by the following variables:

- **testkey** : The final half key as key
- **raw_txt** : The final crypted strin as string.

This is the main Raptor Cryptographic Algorithm v3. It is ruled by the following steps :

- **Initialization** of differents variables and of the Base table via the table generator methods
- **Splitting** part of the given raw string as input. This string will be splitted into differents slices, wiche be crypted one by one and associated to his key via the third level separators wich define the third level of the crypting tree.
- **Crypting procedure** for each of the slices obtained by the split method above. These crypted results will be stored as a list of list, respectively a list of slices, defined by a list of crypted terms.
- **Manage the results** via slurp2 and slurp3 methods. The results are properly stored at this time to be correctly interpreted later.
- **Give a wrong path** for decrypting using some fake values to both of crypted txt and key as strings. It means any Brute force attack will be ignored.
- **Returns** the couple (crypt txt, key) wich is efficient to be decrypted by the solver.

This algorithm is stable in his domain and must be used on it. Please not to try bigger data slice and automate it via shell script if necessary. It should be used as a data crypter using a top level slicer and manager (from the shell script as exemple).

See source below to more explanation.

5.1.2 Source Code

```
import sys
import math as m
import random as r

represent=''
table2 = []
dic = {}
main_dic={}
choice = ' '
chaine=''
chaine=sys.argv[1]

if(len(sys.argv)!=4):
        Basemin = 2
        Basemax = 37
        Range   = 36**2
else :
        Basemin = int(sys.argv[1])
        Basemax = int(sys.argv[2])
        Range   = int(sys.argv[3])

if(Basemin<2 or Basemax>37):
        print("Affichage impossible veuillez selectionner une plage de valeure contenue␣
↪dans [2,36]")
        exit(0)

maxi=Basemax-Basemin
table2=table()
long_chaine = []
long_crypt  = []
longi=0
seuil = 20
seuil_lvl2=70
choice = ''
userchoice=0
sep=['!','"','#','$','%','&','(',')','*','+',',','-','.','/']
sep_lvl2=[":",";","<","=",">","?","@"]
sep_lvl3=['A','B','C','D','E','F','G','H','I','J','K','L']
mesquin=['M','N','O','P','Q','R','S','T','U','V','W','X','Y','Z']

long_long_chaine = []
tmp_long_chaine  = []
long_chaine      = []
long_crypt       = []
testc            = []
testk            = []
int_chaine       = []
lvl2_key_miam    = []
tmp_crypt        = []
```

(continues on next page)

(continued from previous page)

```
while(choice!='q'):
        # init_all()
        current_sep_lvl3     =  "A"
        current_sep_lvl2     =  ":"
        long_chaine[:]       = []
        long_crypt[:]        = []
        long_long_crypt      = []
        testc[:]             = []
        testk[:]             = []
        int_chaine[:]        = []
        lvl2_key_miam[:]     = []
        long_long_chaine[:] = []
        tmp_long_chaine[:]  = []
        tmp_crypt            = ()
        testkey              = ''
        raw_txt              = ''
        clean_txt            = ''
        longi                = 0
        longii               = 0
        res                  = ()

        if(userchoice):
                chaine = ''
                chaine=input("Veuillez entrer la chaine à crypter : ")
        if(len(chaine)>=seuil and len(chaine)<seuil_lvl2):
                long_chaine = split(chaine,seuil)
                longi+=1
        else:
                if(len(chaine)>=seuil_lvl2):
                        tmp_long_chaine = split(chaine,seuil_lvl2)
                        for i in range(0,len(tmp_long_chaine)):
                                long_long_chaine.append(split(tmp_long_chaine[i],seuil))
                        longii+=1
        if(not longi and not longii):
                res=crypt_procedure(chaine,table2)
        else :
                if(longi):
                        for i in range(0,len(long_chaine)):
                                long_crypt.append(crypt_procedure(long_chaine[i],table2))
                if(longii):
                        for i in range (0,len(long_long_chaine)):
                                for j in range(0,len(long_long_chaine[i])):
                                        tmp_crypt = crypt_procedure(long_long_
↪chaine[i][j],table2)
                                        long_long_crypt.append(tmp_crypt)
                        # print(long_crypt[-1][0])
        if(not longi and not longii):
                testc = res[0]
                testk = res[1]
        else :
                if (longi):
                        for i in range (0,len(long_crypt)):
```

(continues on next page)

(continued from previous page)

```
                                for j in range(0,len(long_crypt[i][0])):
                                        testc.append(str(long_crypt[i][0][j]))
                                for k in range(0,len(long_crypt[i][1])):
                                        testk.append(str(long_crypt[i][1][k]))
                                current_sep_lvl2=cyclik_ascii_lvl2(current_sep_lvl2)
                                testc[-1]+=current_sep_lvl2
                                testk[-1]+=current_sep_lvl2
                if(longii):
                        for l in range (0,len(long_long_crypt)):
                                # print(long_long_crypt[l])
                                for j in range(0,len(long_long_crypt[l][0])):
                                        testc.append(str(long_long_crypt[l][0][j]))
                                for k in range(0,len(long_long_crypt[l][1])):
                                        testk.append(str(long_long_crypt[l][1][k]))
                                current_sep_lvl2=cyclik_ascii_lvl2(current_sep_lvl2)
                                testc[-1]+=current_sep_lvl2
                                testk[-1]+=current_sep_lvl2
                                # print("l = "+str(l)+" | len long[l] = "+str(len(long_
↪long_crypt[l][0])))
                                if(len(long_long_crypt[l][0])<seuil):
                                        current_sep_lvl3=cyclik_ascii_lvl3(current_sep_
↪lvl3)
                                        testc[-1]+=current_sep_lvl3
                                        testk[-1]+=current_sep_lvl3
                # print(testc)
                # print(testk)
        int_chaine=(ascii_to_int(chaine))
        for i in range(0,len(testk)):
                testkey+=str(testk[i])
        if(not longi and not longii):
                raw_txt = crypt_final(res,int_chaine,table2)
        else:
                raw_txt += crypt_final_long(testc,int_chaine,table2)
        raw_txt=mesqui(raw_txt,seuil)
        testkey=mesqui(testkey,seuil)
        print("---------------------------------")
        print("Chaine cryptée : \n")
        print(raw_txt)
        print("---------------------------------")
        print("Clé unique : \n")
        print(testkey)
        print("---------------------------------")
        choice=input("c)ontinuer ou q)uitter")
        if(choice!='q'):
                userchoice+=1
```

5.2 Description of De-crypter

Description of the Main Raptor's Cryptographic Algorithm

5.2.1 Algorithm

Description of the Main Raptor's Cryptographic Algorithm

This is the main solver algorithm program. It allow us to decrypt datas slices crypted with the version 1 of the Raptor Cryptographic Algorithm. To solve I need thse following variables :

- **raw_txt** : The input crypted string storage
- **Basemin** : The minimum Base index
- **Basemax** : The maximum Base index
- **table2** : The list of list containing the Base Table
- **testkey** : The key of the algorithm, the decrypting process absolutely need this key.

The solving procedure is ruled by the following steps:

- **Generating the Base Table** and store it into my table2 variable
- **Getting inputs** known as crypted string and his associated key.
- **Organize data slice** removing separators via the slurps methods
- **Decrypting process** using the decrypt_procedure method (see documentation)
- **Store and return** the results of decrypting process

5.2.2 Source Code

```
import sys
import math as m
import random as r

represent=''
table2 = []
dic = {}
main_dic={}
choice = ' '
chaine=''
chaine=sys.argv[1]

if(len(sys.argv)!=4):
        Basemin = 2
        Basemax = 37
        Range   = 36**2
else :
        Basemin = int(sys.argv[1])
```

(continues on next page)

(continued from previous page)

```
        Basemax = int(sys.argv[2])
        Range   = int(sys.argv[3])

if(Basemin<2 or Basemax>37):
        print("Affichage impossible veuillez selectionner une plage de valeure contenue␣
↪dans [2,36]")
        exit(0)

maxi=Basemax-Basemin
table2=table()
long_chaine = []
long_crypt  = []
longi=0
seuil = 20
seuil_lvl2=70
choice = ''
userchoice=0
sep=['!','"','#','$','%','&','(',')','*','+',',','-','.','/']
sep_lvl2=[":",";","<","=",">","?","@"]
sep_lvl3=['A','B','C','D','E','F','G','H','I','J','K','L']
mesquin=['M','N','O','P','Q','R','S','T','U','V','W','X','Y','Z']

long_long_chaine = []
tmp_long_chaine  = []
long_chaine      = []
long_crypt       = []
testc            = []
testk            = []
int_chaine       = []
lvl2_key_miam    = []
tmp_crypt        = []


while(choice!='q'):
        # init_all()
        current_sep_lvl3     =  "A"
        current_sep_lvl2     =  ":"
        long_chaine[:]       = []
        long_crypt[:]        = []
        long_long_crypt      = []
        testc[:]             = []
        testk[:]             = []
        int_chaine[:]        = []
        lvl2_key_miam[:]     = []
        long_long_chaine[:]  = []
        tmp_long_chaine[:]   = []
        tmp_crypt            = ()
        testkey              = ''
        raw_txt              = ''
        clean_txt            = ''
        longi                = 0
        longii               = 0
```

(continues on next page)

(continued from previous page)

```
        res                   = ()

        raw_txt=input("Veuillez entrer la chaine cryptée : \n")
        testkey=input("Veuillez saisir la clé : \n")
        if(len(raw_txt)>=seuil*6 and len(raw_txt)<seuil_lvl2*6):
                long_chaine = split(raw_txt,seuil)
                longi+=1
        else:
                if(len(raw_txt)>=seuil_lvl2*6):
                        tmp_long_chaine = split(raw_txt,seuil_lvl2*6)
                        for i in range(0,len(tmp_long_chaine)):
                                long_long_chaine.append(split(tmp_long_chaine[i],seuil))
                        longii+=1

        raw_txt = slurp3(raw_txt)
        testkey = slurp3(testkey)
        if(not longi and not longii):
                clean_txt = decrypt_procedure(raw_txt,testk,table2)
        else:
                if(longi):
                        lvl2_liste = []
                        lvl2_key   = []
                        lvl2_liste = slurp2(raw_txt)
                        lvl2_key   = slurp2(testkey)
                        lvl2_key_miam = []
                        # print(lvl2_liste)
                        # print(lvl2_key)
                        for i in range (0,len(lvl2_key)):
                                lvl2_key_miam.append(miam(lvl2_key[i]))
                        # print(lvl2_key_miam)
                        for i in range (0,len(lvl2_liste)-1):
                                clean_txt+= decrypt_procedure(lvl2_liste[i],lvl2_key_
↪miam[i],table2)
                if(longii):
                        lvl3_liste = []
                        lvl3_key   = []
                        lvl3_liste = slurp4(raw_txt)
                        lvl3_key   = slurp4(testkey)
                        lvl2_liste = []
                        lvl2_key   = []
                        lvl2_key_miam = []
                        final_key  = []
                        for i in range (0,len(lvl3_key)):
                                lvl2_key.append(slurp2(lvl3_key[i]))
                        for i in range (0,len(lvl3_liste)-1):
                                lvl2_liste.append(slurp2(lvl3_liste[i]))
                        for i in range(0,len(lvl2_key)-1):
                                lvl2_key_miam[:] = []
                                for j in range (0,len(lvl2_key[i])):
                                        lvl2_key_miam.append(miam(lvl2_key[i][j]))
                                        # print("miam")
                                        # print(lvl2_key_miam)
```

(continues on next page)

(continued from previous page)

```
                                del lvl2_key_miam[-1]
                                final_key.append(lvl2_key_miam)
                                # print("final")
                                # print(final_key)
                                # print("liste : "+str(len(lvl2_liste))+" | key
↪"+str(len(final_key)))
                                for k in range (0,len(lvl2_liste[i])-1):
                                        # print("lvl2[i][k] : ")
                                        # print(lvl2_liste[i][k])
                                        # print(final_key[0][k])
                                        clean_txt+=decrypt_procedure(lvl2_liste[i][k],
↪final_key[0][k],table2)
                                        # print(str(k) + "/" + str(len(lvl2_liste[i])-2))
                                # print(str(i)+" / "+str(len(lvl2_key)-1))

        print("Chaine décryptée : \n")
        print(clean_txt)
        choice=input("c)ontinuer ou q)uitter")
        if(choice!='q'):
                userchoice+=1
```

5.3 ascii_to_int

```
def ascii_to_int(chaine)
```

5.3.1 Algorithm

Utils method : ascii to integer converter.

Parameters	Type	Description
chaine	*str*	The string to convert

Returns **list** : A list containing all integers values since ASCII.

5.3.2 Source Code

```
res = []
for letter in chaine:
        res.append(ord(letter))
return res
```

5.4 int_to_ascii

```
def int_to_ascii(crypt)
```

5.4.1 Algorithm

Utils method : integer to ascii converter.

Description	Type	Description
crypt	*int list*	The int list to convert

Returns str : The converted ASCII string since int list.

5.4.2 Source Code

```
res = ''
for i in range (0,len(crypt)):
        res+=chr(crypt[i])
return res
```

5.5 cryptChaine

```
def cryptChaine(to_crypt,table,base)
```

5.5.1 Algorithm

The simple method to crypt an ascii string as integer list.

Parameters	Type	Description
to_crypt	*int list*	The converted int list since an ascii string
table	*list of list*	An array containing all fully builded Base Table
base	*int*	Define the Base index

Returns **str list** : A string list containing all the base crypted values. Must be used as a crypted list.

5.5.2 Source Code

```
res = []
for i in range(0,len(to_crypt)):
        res.append(table[base][to_crypt[i]])
return res
```

5.6 local_table_dico

```
def local_table_dico(table2,base,rangeB)
```

5.6.1 Algorithm

Utils method : A method to convert a Base table to Python dictionnary

Parameters	Type	Description
table2	*list of list*	An array containing all the fully builded Base table
base	*int*	Define the Base index
rangeB	*int*	Define the max step of incrementation

Returns Dictionnary : A dictionnary representing the specified Base table

5.6.2 Source Code

```
str_base={}
res = {}
if(rangeB>base**2):
        rangeB=base**2
for i in range (0,rangeB):
        str_base[i]=table2[base][i]
return str_base
```

5.7 limit_range

```
def limit_range(Range,base)
```

5.7.1 Algorithm

Utils method : A method to limit the Base range

Parameters	Type	Description
Range	*int*	The range as a limit
base	*int*	The current Base index

Returns **int** : The limited by range res.

5.7.2 Source Code

```
res=0
if(Range>base**2):
        res=base**2
else:
        res=Range
return res
```

5.8 base_key

```
def base_key(int_chaine)
```

5.8.1 Algorithm

This is the key builder.

Parameters	Type	Description
int_chaine	*int list*	The base index list as a starting builder for key

Returns **int list** : the builded key from index base list.

5.8.2 Source Code

```
res=[]
for i in range (0,len(int_chaine)):
        tmp=((int_chaine[i]*int_chaine[len(int_chaine)-i-1]+10)%36)
        if(tmp<10):
                tmp+=10
        res.append(tmp)
return res
```

5.9 vec_poids

```
def vec_poids(int_chaine)
```

5.9.1 Algorithm

Compute the vectorial cumulated weight of the list.

Parameters	Type	Description
int_chaine	*int list*	The integer list to treat

Returns int list : The computed accumulated weigth integer list

5.9.2 Source Code

```
res = []
res.append(int_chaine[0])
for i in range(1,len(int_chaine)):
        res.append(res[i-1]+int_chaine[i])
return res
```

5.10 vec_1_poids

```
def vec_1_poids(vec_poids)
```

5.10.1 Algorithm

Compute the inverse of the vectorial cumulated weigth computation.

Parameters	Type	Description
vec_poids	*int list*	The weigth as an integer list

Returns int list : The computed list containing the inverse operation of vec_poids method

5.10.2 Source Code

```
res=[]
for i in range (0,len(vec_poids)):
        res.append(1/vec_poids[i])
return res
```

5.11 equa_2_nd

```
def equa_2_nd(a,b,c)
```

5.11.1 Algorithm

Utils : An 2nd order equation solver

Parameters	Type	Description
a	*int / float*	The a coefficient
b	*int / float*	The b coefficient
c	*int / float*	The c coefficient

Returns int / float : The solved equation positive root

5.11.2 Source Code

```
res = 0
racine1 = 0.0
racine2 = 0.0
delta = b**2-4*a*c
if(delta>0):
        racine1 = (-b+m.sqrt(delta))/2*a
        racine2 = (-b-m.sqrt(delta))/2*a
if(racine1>0):
        res = int(racine1)
else:
        res = int(racine2)
return res
```

5.12 multlist

```
def multlist(a,b)
```

5.12.1 Algorithm

Utils : A point by point list multiplier

Parameters	Type	Description
a	*int/float list*	The list to multiply
b	*int/float list*	The list to multiply

Returns int / float list : The computed point by point multiplication

5.12.2 Source Code

```
res = []
if(len(a)!=len(b)):
        return []
else:
        for i in range(0,len(a)):
                res.append(a[i]*b[i])
return res
```

5.13 transpose_base

```
def transpose_base(liste,key,table)
```

5.13.1 Algorithm

A method to transpose an integer list to the corresponding key's base index => The result will be a succession of transposed values from differents integers to differents base

Parameters	Type	Description
liste	*list*	the integer converted since ASCII list
key	*list*	The Base index list as key
table	*list*	The full Base Table recursively builded

Returns **str list** : The crypted list as String list

5.13.2 Source Code

```
res = []
if(len(liste)!=len(key)):
        return []
else :
        for i in range (0,len(liste)):
                if(key[i]==10):
                        res.append(liste[i])
                else:
                        res.append(table[key[i]-2][liste[i]])
return res
```

5.14 inv_transpose_base

```
def inv_transpose_base(liste,key,table)
```

5.14.1 Algorithm

The inverse method to decrypt a str list of base transposed values

Parameters	Type	Description
liste	*str list*	The crypted list as String list
key	*int list*	The Base index list as key
table	*int list*	The full Base table recursively builded

Returns **int list** : The decrypted list as integers

5.14.2 Source Code

```
res = []
if(len(liste)!=len(key)):
        return []
else:
        for i in range(0,len(liste)):
                if(key[i]==10):
                        res.append(int(liste[i]))
                else:
                        res.append(int(table[key[i]-2].index(liste[i])))
return res
```

5.15 crypt_procedure

```
def crypt_procedure(chaine,table)
```

5.15.1 Algorithm

The crypter manager to orchestrate the crypting procedure. It works from these steps:

- *We convert the given ascii string as integer list*
- *We compute the Base index list as key from the converted integer list*
- *We build the second part of the key since the mirror of the Base index list*
- *We compute the cumulated weight of the integer list*
- *We compute the point by point multiplication between cumulated weigth list and original integer list*
- *We transpose the multiplied list into the given specified Base from the key*
- *We associate the crypted strin to the key as return*

Parameters	Type	Description
chaine	*string*	The string to crypt
table	*list of list*	The Base Table recursively builded

Returns **list tuple** : The couple crypted string and key as result. It permits to decrypt any message.

5.15.2 Source Code

```
int_chaine = ascii_to_int(chaine)
base_keyy  = base_key(int_chaine)
if(len(base_keyy)%2==0):
        key=base_keyy[0:int(len(base_keyy)/2)]
else:
        key=base_keyy[0:int((len(base_keyy)/2)+1)]
vec_poid   = vec_poids(int_chaine)
crypt_lst  = multlist(int_chaine,vec_poid)
crypt_lst  = transpose_base(crypt_lst,base_keyy,table)
# print(crypt_lst)
return(crypt_lst,key)
```

5.16 cyclik_ascii

```
def cyclik_ascii(current)
```

5.16.1 Algorithm

Compute a cyclik ascii separators into ponctuation signs

Parameters	Type	Description
current	*str*	The current poncuation separator

Returns **str** : The following separator from the defined 'sep' Set.

5.16.2 Source Code

```
sep=['!','"','#','$','%','&','(',')','*','+',',','-','.','/']
tmp=((sep.index(current)+1)%13)
res =sep[tmp]
return res
```

5.17 cyclik_ascii_lvl2

```
def cyclik_ascii_lvl2(current)
```

5.17.1 Algorithm

Compute a cyclik ascii separators into ponctuation signs. Get a second cyclic ascii set modulo length

Parameters	Type	Description
current	*str*	The current poncuation separator

Returns **str** : The following separator from the defined 'sep' Set.

5.17.2 Source Code

```
sep=[":",";","<","=",">","?","@"]
tmp=((sep.index(current)+1)%6)
res =sep[tmp]
return res
```

5.18 cyclik_ascii_lvl3

```
def cyclik_ascii_lvl3(current)
```

5.18.1 Algorithm

Compute a cyclik ascii separators into Upper letters from A to L. Get a third cyclic ascii set modulo length

Parameters	Type	Description
current	*str*	The current poncuation separator

Returns **str** : The following separator from the defined 'sep' Set.

5.18.2 Source Code

```
sep=['A','B','C','D','E','F','G','H','I','J','K','L']
tmp=r.randint(0,11)
res = sep[tmp]
return res
```

5.19 cyclik_ascii_mesquin

```
def cyclik_ascii_mesquin(current,int_chaine)
```

5.19.1 Algorithm

Compute a cyclik ascii separators into Upper letters from M to Z. Get a third cyclic ascii set modulo length

Parameters	Type	Description
current	*str*	The current poncuation separator

Returns **str** : The following separator from the defined 'sep' Set.

5.19.2 Source Code

```
mesquin=['M','N','O','P','Q','R','S','T','U','V','W','X','Y','Z']
tmp=r.randint(0,11)
res=mesquin[tmp]
return res
```

5.20 reverse

```
def reverse(liste)
```

5.20.1 Algorithm

A function to reverse a string as argument.

Parameter	Type	Description
s	*String*	The string to reverse

Returns **str** : The reversed string

5.20.2 Source Code

```
res=[]
for i in range(0,len(liste)):
        res.append(liste[len(liste)-i-1])
return res
```

5.21 split_number

```
def split_number(num)
```

5.21.1 Algorithm

Integer splitter using the inverse Horner scheme and get it as a list of digits.

Parameters	Type	Description
num	*int*	The integer to be splitted

Returns **list** : The splitted integer as list

5.21.2 Source Code

```
res=[]
while(num>0):
        res.append(num % 10)
        num=int(num/10)
return reverse(res)
```

5.22 complement_at

```
def complement_at(x,base=2)
```

5.22.1 Algorithm

Get the direct Base complemented value from the original x value. The Base must be inferior or equal to 10.

Parameters	Type	Description
x	*int*	The value to be complemented
base	*int*	The current base

Returns **int** : The complemented value as an integer

5.22.2 Source Code

```
return (base-1-x)
```

5.23 get_value

```
def get_value(x,table,base)
```

5.23.1 Algorithm

A value getter to obtain an index from the original Base converted string value. This method is working as the list 'index' method and allow us to get the raw full integer corresponding to the list of list value.

Parameters	Type	Description
x	*str*	The value to search
table	*list of list*	The full Base Table
base	*int*	The index of the base

Returns int : The real decimal value of the specified term in his own Base.

5.23.2 Source Code

```
ind=0
while(table[base][ind]!=x):
        ind+=1
return ind
```

5.24 complement_at_sup11

```
def complement_at_sup11(x,table,base=11)
```

5.24.1 Algorithm

This function is used to compute the complement value from the original one in his own base. I use a temporary variable to store the numeric value of the compement and restitute it in his own base.

Parameters	Type	Description
x	*str*	A string representation of my base converted value
table	*list of list*	The full Base Table array
base	*int*	The base index of the current value

Returns **str** : The complmented value in his own Base.

5.24.2 Source Code

```
nb_char=len(x)
local_max=0
for i in range(0,nb_char):
        local_max+=(base-1)*base**i
num_value=local_max-get_value(x,table,base)
return table[base][num_value]
```

5.25 complement

```
def complement(x,table,base=2)
```

5.25.1 Algorithm

The complement function is the full algorithm combining the complement_at_sup11 and complement_at functions. I specify the way to take between both of them using an if then else structure.

Parameters	Type	Description
x	*str*	A string representation of my base converted value
table	*list of list*	The full Base Table array
base	*int*	The base index of the current value

Returns str : The complmented value in his own Base.

5.25.2 Source Code

```
final_res=0
if(base<=10):
        splitted=split_number(int(x))
        for i in range(0,len(splitted)):
                splitted[i]=complement_at(splitted[i],base)
                final_res*=10
                final_res+=splitted[i]
        return final_res
else:
        final_res=complement_at_sup11(x,table,base)
        return final_res
```

5.26 crypt_final

```
def crypt_final(tuple,int_chaine,table)
```

5.26.1 Algorithm

The layout procedure to organise crypting results. The uodate consist to complement each of terms in his corresponding base. It allow a superior level of crypting. I use the separators set as well.

Parameters	Type	Description
tuple	*tuple*	List couple representing the crypted strin and the associated key

Returns **str** : The crypted list as a string with correct separators

5.26.2 Source Code

```
sept=['!','"','#','$','%','&','(',')','*','+',',','-','.','/']
res = ''
sep =sept[int(int_chaine[1]*m.cos(int_chaine[0]))%13]
crypt=tuple[0]
key=tuple[1]
tmp_len=len(key)
if(len(key)%2==0):
        for i in range(1,tmp_len):
                key.append(key[tmp_len-i-1])
else:
        for i in range(0,tmp_len):
                key.append(key[tmp_len-i-1])
for i in range (0,len(crypt)):
        # injective crypt[i]
        res+=sep+str(complement(crypt[i],table,key[i]))
        sep=cyclik_ascii(sep)
return res
```

5.27 crypt_final_long

```
def crypt_final_long(liste,int_chaine,table)
```

5.27.1 Algorithm

Chaining the final-level algorithm to get complex crypto-procedure

Parameters	Type	Description
tuple	*tuple*	List couple representing the crypted string and the associated key

Returns **str** : The full second level crypted string

5.27.2 Source Code

```
sept=['!','"','#','$','%','&','(',')','*','+',',','-','.','/']
res = ''
sep =sept[int(int_chaine[1]*m.cos(int_chaine[0]))%13]
for i in range (0,len(liste)):
        res+=sep+str(liste[i])
        sep=cyclik_ascii(sep)
# print(res)
return res
```

5.28 slurp

```
def slurp(chaine)
```

5.28.1 Algorithm

This method allow us to rebuild a str list of crypted terms using separators set.

Parameters	Type	Description
chaine	*str*	The raw string crypted message

Returns str list : The list of crypted terms rebuilded from the raw string

5.28.2 Source Code

```
tmp=''
res = []
sep=['!','"','#','$','%','&','(',')','*','+',',','-','.','/']
for elem in chaine:
        if(not elem in sep):
                tmp+=str(elem)
                # print("tmp = "+tmp)
        else :
                res.append(tmp)
                # print("res = ")
                # print(res)
                tmp=''
        if(elem==''):
                break
res=res[1:]
res.append(tmp)
return res
```

5.29 slurp2

```
def slurp2(chaine)
```

5.29.1 Algorithm

This method is similar of the slurp method. It defined a second level of crypting management.

Parameters	Type	Description
chaine	*str*	The raw string crypted message

Returns **str list** : The list of crypted terms rebuilded from the raw string.

5.29.2 Source Code

```
tmp=''
res = []
sep=[":",";","<","=",">","?","@"]
for elem in chaine:
        if(not elem in sep):
                tmp+=str(elem)
        else:
                res.append(tmp)
                tmp=''
        if(elem==''):
                break
res.append(tmp)
return res
```

5.30 slurp3

```
def slurp3(chaine)
```

5.30.1 Algorithm

This method is similar of the slurp2 method. It defined a third level of crypting management.

Parameters	Type	Description
chaine	*str*	The raw string crypted message

Returns str list : The list of crypted terms rebuilded from the raw string.

5.30.2 Source Code

```
tmp=''
mesquin=['M','N','O','P','Q','R','S','T','U','V','W','X','Y','Z']
for elem in chaine:
        if(not elem in mesquin):
                tmp+=str(elem)
return tmp
```

5.31 slurp4

```
def slurp4(chaine)
```

5.31.1 Algorithm

This method is similar of the slurp2 method. It defined a third level of crypting management.

Parameters	Type	Description
chaine	*str*	The raw string crypted message

Returns str list : The list of crypted terms rebuilded from the raw string.

5.31.2 Source Code

```
tmp=''
res = []
sep=['A','B','C','D','E','F','G','H','I','J','K','L']
for elem in chaine:
        if(not elem in sep):
                tmp+=str(elem)
        else:
                res.append(tmp)
                tmp=''
        if(elem==''):
                break
res.append(tmp)
return res
```

5.32 miam

```
def miam(key)
```

5.32.1 Algorithm

Key builder from the half key as integer list. It rebuild the missing half with a mirror copy of the first one.

Parameters	Type	Description
key	*int list*	The half key as int list

Returns int list : The full key rebuilded from the half key

5.32.2 Source Code

```
tmp=''
count=1
res=[]
for this in key:
        # print("this = "+str(this))
        # print("tmp = "+str(tmp))
        if(count%2==0):
                tmp+=str(this)
                count=1
                # print("tmp = "+str(tmp))
                res.append(tmp)
                tmp=''
        else:
                tmp=str(this)
                count+=1
for i in range(0,len(res)):
        res[i]=int(res[i])
return res
```

5.33 resolve

```
def resolve(liste)
```

5.33.1 Algorithm

This method compute the chained 2nd order equations to solve the numeric suit. It permit us to get the ASCII values as a list. To solve the system you have to instance the solver with the square root of term 0. Once theorem zero done, you will apply the equation solver with square root of the 0-term as b, a as 1 and c as -following term. The algorithm sort the roots and take only positives ones.

Parameters	Type	Description
liste	*int list*	The computed multiplied list to solve

Returns **int list** : A list containing solved terms.

5.33.2 Source Code

```
res = []
x = 0
tmp2 = 0
res.append(int(m.sqrt(liste[0])))
tmp=res[0]
for i in range (1,len(liste)):
        # print("y = "+str(tmp))
        # print("x = "+str(x))
        tmp2 = equa_2_nd(1,-tmp,-liste[i])
        x=tmp2-tmp
        res.append(int(x))
        tmp=tmp2
# print(res)
return res
```

5.34 decrypt_procedure

```
def decrypt_procedure(chaine,key,table)
```

5.34.1 Algorithm

This method manage the decrypting procedure. It is ruled by the following steps :

- Build the full key since the key argument
- Split the string since separators via slurp method
- Complement eah ch term in his own value
- Apply the inv_tranpose_base method to get the uncrypted terms
- Solve the cumulated multiplued weigth with the equation solver
- Convert the int list as result to ASCII chain

Parameters	Type	Description
chaine	*str*	The raw crypted text as string
key	*int list*	The half key as int list
table	*list of list*	The Base Table array

Returns **str** : The uncrypted text.

5.34.2 Source Code

```
res = ''
base=key[:]
tmp = []
key.reverse()
tmp = key[:]
to_find = []
to_find=slurp(chaine)
print(len(to_find))
print(len(key))
for i in range(0,len(to_find)):
        #injective inverse to_find[i]
        to_find[i]=complement(to_find[i],table,base[i])
tmp_liste=inv_transpose_base(to_find,base,table)
int_liste=resolve(tmp_liste)
res = int_to_ascii(int_liste)
return res
```

5.35 split

```
def split(chaine,seuil)
```

5.35.1 Algorithm

Split the given string argument 'chaine' into slices from threshold size 'seuil'. Each of this slices are allowed into the cryptographic algorithm.

Parameters	Type	Description
chaine	*str*	The full string to treat
seuil	*int*	Define the threshold size of the slices

Returns **str list** : The slices list as result

5.35.2 Source Code

```
res = []
tmp = ''
index = 0
div=int(len(chaine)/seuil)
for i in range(0,div):
        tmp=''
        # print("index = "+str(index)+" | seuil = "+str(seuil)+" | i = "+str(i))
        for j in range(index,(index+seuil)):
                tmp+=chaine[j]
                # print("j = "+str(j)+" | tmp = "+str(tmp))
                if(j==(index+seuil-1)):
                        index=j+1
        res.append(tmp)
if((index-1)<len(chaine)):
        tmp=chaine[index:]
        res.append(tmp)
return res
```

5.36 tilps

```
def tilps(chaine)
```

5.36.1 Algorithm

The reverse method of the split function. From a given str list, we rebuild the full length string

Parameters	Type	Description
chaine	*str list*	The String slices as a list

Returns **str** : The full striing rebuilded from the slices list

5.36.2 Source Code

```
res = ''
for i in range (0,len(chaine)):
        res+=chaine[i]
return res
```

5.37 mesqui

```
def mesqui(txt,seuil)
```

5.37.1 Algorithm

This method is used to create a wrong path of decrypting method. Using a similar Separators terms, I define a 'fake' terms list wich have absolutely no meanings for the rest of the algorithm. Using it as the last step of algorithm, it doesn't allow any brute force attack to decrypt. The threshold value 'seuil' will define the amount of distribution of fake separators.

Parameters	Type	Description
txt	*str*	The raw string to treat
seuil	*int*	The threshold variable to assign the 'fake terms' length

Returns str : The fully 'fake splitted' crypted string

5.37.2 Source Code

```
mesquin=['M','N','O','P','Q','R','S','T','U','V','W','X','Y','Z']
res=''
sep='M'
for i in range(0,len(txt)):
        res+=txt[i]
        if(i%int((seuil))==0):
                res+=sep
                sep=cyclik_ascii_mesquin(sep,int_chaine)
return res
```

CHAPTER

SIX

RAPTOR CRYPTOGRAPHIC ATERNATIVE ALGORITHM V1

6.1 Description of Crypter

Main Raptor Cryptographic Alternative Algorithm

6.1.1 Algorithm

This is the main Raptor Cryptographic Alternative algorithm. During my researches, I have thought about an other version of the algorithm optimised for the long data stream as string. The first algorithm use exponentional integer values list instead of this one wich allow to treat bigger slices using a divider. Each term will be divide during the algorithm. This algorithm is rules by following steps :

- **Getting inputs**
- **Converting ASCII** to Integers values to get a numeric list
- **Dividing chain** : eachterm is divided by the next one
- **Multiplying** each i_term to the i+1_term modulo the i+2_term to get the key modulo 26. It means key(i)=((data(i)*data(i+1)) modulo data(i+2)) modulo 26
- **Multiplying** each term of the crypt list with 10000 to get integers values from float.
- **Key padding** to confirm key appending the two first elements of the key at the end and the top one numric list at the end
- **Transposing** to the associated key index Base the full data list
- **Adding separators** from the sep Set to split each term from another

---

6.1.2 Source Code

```
from base_opt import *
import random as r

sep=['!','"','#','$','%','&','(',')','*','+',',','-','.','/']
vir=[]

# Construction de la table des bases

table=table()

# Algorithme de cryptage

txt=input("Entrez un texte")
l=[]
res=[]
for i in range(len(txt)):
        l.append(ord(txt[i]))

first=int(l[0])
for i in range(0,len(l)-1):
```

(continues on next page)

(continued from previous page)

```
        res.append(float(l[i+1]/l[i]))
key=[]
for i in range(0,len(l)-2):                # Finir la chaine de texte par trois␣
↪caractères "usuels", par exemple "..."
        key.append(int((l[i]*l[i+1])%l[i+2])) # Eventuellement ameliorer la clé en la␣
↪complementant a 36 sur [10,36]
        key[i]=(key[i])%26

for i in range(len(res)):
        res[i]=int(res[i]*10000)
res.append(first)
key.append(key[0])      #key padding
key.append(key[1])
key.append(first)

crypt=[]
for i in range(len(res)):
        crypt.append(table[key[i]][res[i]])

# rajouter des operations de listes reversibles

string=""
for i in range(len(crypt)):
        string+=crypt[i]
        string+=sep[r.randint(0,13)]

str_key=''
for i in range(len(key)):
        str_key+=str(key[i])
        str_key+=sep[r.randint(0,13)]

print(str_key)
# print("################################")
print('!'+string)
# print("################################")
quit()
```

6.2 Description of De-Crypter

Main Raptor Cryptographic Alternative Algorithm

6.2.1 Algorithm

To decrypt the obtained string sequence from the Crypter, you have to follow these steps :

- **Rebuild** original term list from the string using the sep Set
- **Transpose** each term in his corresponding Base from the key to get integers values.
- **Dividing** each of term by 10000 to restitute float values
- **The zero step of decrypting** is the multiplication of the first term of the list with the first value of the begining Ascii converted list (appending it to the key to make it confidential)
- **Restitute** each i_term multiplying with i-1_term
- **Rounding and restitute** via conversion the origial ASCII chain.

6.2.2 Source Code

```
from base_opt import *
import random as r
sep=['!','"','#','$','%','&','(',')','*','+',',','-','.','/']
vir=[]

# Construction de la table des bases

table=table()


string=input('chaine cryptée : ')
str_key=input('clé : ')

key=[]
tmp=''
ind=0
for item in str_key:
	if not item in sep:
		tmp+=item
	else:
		key.append(int(tmp))
		tmp=''
		ind+=1



rez=[]
tmp=''
```

(continues on next page)

(continued from previous page)

```
ind=0
first = key[-1]
key= key[:-1]
for item in string:
        if not item in sep:
                tmp+=item
                # print(tmp)
        else:
                rez.append(table[key[ind]].index(tmp))
                tmp=''
                ind+=1
firstt=rez[-1]
rez=rez[:-1]

# Algorithme de décryptage

for i in range(0,len(rez)):
        rez[i]=rez[i]/10000

rezz=[]
rezz.append(first*rez[0])

for i in range(1,len(rez)):
        rezz.append(rez[i]*rezz[i-1])

final=[]

for i in range(len(rezz)):
        final.append(chr(round(rezz[i])))
txt=''
txt=(chr(first))
for i in range(len(final)):
        txt+=str(final[i])

print(txt)
```

CHAPTER

SEVEN

RAPTOR CRYPTOGRAPHIC ATERNATIVE ALGORITHM V2

7.1 Description of Crypter

Main Raptor Cryptographic Alternative Algorithm

7.1.1 Algorithm

This is the main Raptor Cryptographic Alternative algorithm v2. The difference between both versions is the type of the numbers list. The second version is using a representation of float crypted preserving the full precision of the values. This one is stable on his definition's domain and could be considered as the first one as 'fast crypting algorithm'. There are differents ways to use :

- **Cybersecurity of business and organization** (Hospitals, banks, etc)
- **Crypting data stream** on the web
- **Crypting authentification** informations

This algorithm is ruled by the followings steps :

- **Define two differents sets** :
  - **The sep Set** representing terms separators
  - **The vir Set** reprseneting the comma in float values
- **Getting raw string** as input
- **Converting ASCII values** to their decimal correspondence
- **Dividing each i+1_term of the list by the i_term** of the list
- **Building key** from the given formula : key(i)=((l(i)*l(i+1) modulo l(i+2)) modulo 26)
- **Multiplying each term by 10000**
- **Building the mirror key** from the original one
- **Compute each fraction** division float value. Each fraction is defined by res(i)/key(i+1). Each part of the value is represented into a single integer value
- **Multiplying each float res by 10** to get larger values (useful to Base Table converter)
- **Convert** into key-indexed Base Table values
- **Defining commas and separators** from the vir and sep Sets
- **Return the full crypted string**

7.1.2 Source Code

```
from base_opt import *
import random as r
sep=['!','"','#','$','%','&','(',')','*','+',',','-','.','/']
vir=['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']
```

(continues on next page)

(continued from previous page)

```
table=table()

# Algorithme de cryptage

txt=input("Entrez un texte")
l=[]
res=[]
for i in range(len(txt)):
        l.append(ord(txt[i]))

first=int(l[0])
for i in range(0,len(l)-1):
        res.append(float(l[i+1]/l[i]))
key=[]
for i in range(0,len(l)-2):                # Finir la chaine de texte par trois␣
↪caractères "usuels", par exemple "..."
        key.append(int((l[i]*l[i+1])%l[i+2])) # Eventuellement ameliorer la clé en la␣
↪complementant a 36 sur [10,36]
        key[i]=(key[i])%26

for i in range(len(res)):
        res[i]=int(res[i]*10000)
res.append(first)
key.append(key[0])      #key padding
key.append(key[1])

tmp=0
Float_res=[]
Mirror_key=[]
Mirror_key=mirror(key)
Mirror_key.append(first)
# Compute each fraction division float value. Each fraction is defined by res(i)/key(i+1)
# Each part of the value is represented into a single integer value
for i in range(len(res)):
        tmp = res[i]/(key[i]+1)
        Float_res.append(int(tmp))
        Float_res.append(int((tmp-int(tmp))*1000))
        tmp=0.0

#Multiplying each float res by 10 to get larger values (useful to Base Table converter)
for i in range(len(Float_res)):
        Float_res[i]*=10

#Convert into key-indexed Base Table values
crypt=[]
for i in range(len(Float_res)):
        crypt.append(table[Mirror_key[i]][Float_res[i]])

# rajouter des operations de listes reversibles

string=""
ind=0
```

(continues on next page)

(continued from previous page)

```
# Defining commas and separators from the vir and sep Sets
for i in range(len(crypt)):
        string+=crypt[i]
        if(ind%2==0):
                string+=vir[r.randint(0,25)]
        else:
                string+=sep[r.randint(0,13)]
        ind+=1

ind=0
str_key=''
for i in range(len(Mirror_key)):
        str_key+=str(Mirror_key[i])
        str_key+=sep[r.randint(0,13)]

print("key : ")
print(str_key)
# Return the full crypted string
print("################################")

print("string = ")
print(string)
print("################################")
quit()
```

7.2 Description of De-Crypter

Main Raptor Cryptographic Alternative Algorithm

7.2.1 Algorithm

To decrypt the obtained string sequence from the Crypter, you have to follow these steps :

- **Rebuild the terms list** from the given string using sep and vir Sets
- **Convert crypted value** to their integer index
- **Devide each of value by 10** to get the smaller origianl values
- **Rebuild float values** from the integer couples values
- **Round multiplication** of float value and Mirror key value to rebuild terms
- **Divide each computed values** from multiplication of i_term fo the float list with the last computed term by 10000 to get origianls terms
- **Round and convert to ASCII** values to get the original string

---

7.2.2 Source Code

```
from base_opt import *
import random as r

sep=['!','"','#','$','%','&','(',')','*','+',',','-','.','/']
vir=['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']

table=table()


string=input('Chaine Cryptée : ')
str_key=input('Clé : ')

Mirror_key=[]
tmp=''
ind=0
for item in str_key:
        if not item in sep :
                tmp+=item
        else:
                Mirror_key.append(int(tmp))
                tmp=''
                ind+=1

rez=[]
```

(continues on next page)

(continued from previous page)

```
tmp=''
ind=0
first=Mirror_key[-1]
Mirror_key=Mirror_key[:-1]

# Rebuild the terms list from the given string using sep and vir Sets
for item in string:
        if not item in sep and not item in vir:
                tmp+=item
        else:
                # Convert crypted value to their integer index
                rez.append(table[Mirror_key[ind]].index(tmp))
                tmp=''
                ind+=1
firstt=rez[-1]
rez=rez[:-1]

# Devide each of value by 10 to get the smaller origianl values
for i in range(len(rez)):
        rez[i]/=10
Float_rez=[]

# Rebuild float values from the integer couples values
for i in range(0,len(rez)):
        if(i%2==0):
                tmp=rez[i]
        else:
                Float_rez.append(tmp+rez[i]/1000)
                tmp=0.0

# Algorithme de décryptage

# Round multiplication of float value and Mirror key value to rebuild terms
rez=[]
for i in range(len(Float_rez)):
        rez.append(round(Float_rez[i]*(Mirror_key[i]+1)))

rezz=[]
rezz.append((first*rez[0])/10000)

# Divide each computed values from multiplication of i_term fo the float list with the␣
↪last computed term by 10000 to get origianls terms
for i in range(1,len(rez)):
        rezz.append((rez[i]*rezz[i-1])/10000)

final=[]
# Round and convert to ASCII values to get the original string
for i in range(len(rezz)):
        final.append(chr(round(rezz[i])))
txt=""
txt=(chr(first))
for i in range(len(final)):
```

(continues on next page)

(continued from previous page)

```
        txt+=final[i]

print(txt)
```

7.3 mirror

```
def mirror(liste)
```

7.3.1 Algorithm

The mirror function build a mirror list from the given one.

Parameters	Type	Description
liste	list	The list to be treat

Returns list : The mirror list from the given parameter.

7.3.2 Source Code

```
res=liste[:]
for i in range(1,len(liste)):
        res.append(liste[-i])
res.append(liste[0])
return res
```

CHAPTER

EIGHT

MATH PROOF

8.1 Crypting Protocol

We will crypt a simple message containing the word 'salut'.

In a first step we have to compute the weight list of the differents caracters (meaning an approximation of the ASCII code used in the computer code algorithm).

8.1.1 Weigth List

Giving 0 to 'a' to 26 to 'z', we have : *18.0.11.20.19* as the weigth list of the string

8.1.2 Cumulated weigth list

Once done, we have to compute the cumulated weigth list. I mean, the list application can be considered as a suit defined by :

u_n a suit from N to N with the length $n \in$ N | u_i=u_{i-2}+u_{i-1}

In our case, the computed list is *18.18.29.49.68* We call it v_i

8.1.3 Key Computing

At this moment we have to compute the public key k_i of the algorithm defined via modulo since the formula :

$$\begin{cases} k_i = [u_i.u_{n-i} \quad mod \quad 26] + 10 & if \quad \exists u_i, u_{n-i} \\ u_j & if \quad !\exists u_j, j = n/2 + 1 \end{cases}. \tag{8.1}$$

With our example, it gives :

$$\begin{cases} k_0 = [18.19 \quad mod \quad 26] + 10 = 14 \\ k_1 = [20.0 \quad mod \quad 26] + 10 = 10 \\ k_2 = [11 \quad mod \quad 26] + 10 = 11 \end{cases}. \tag{8.2}$$

We build the full Length key ξ using the formula :

$$\begin{cases} \xi_i = k_i & if \quad i <= n/2 + 1 \\ \xi_i = k_{n-i} & if \quad i > n/2 + 1 \end{cases}. \tag{8.3}$$

8.1.4 Crypting Process

The crypting process is ruled by a pseudo-convolution with the given symbol $*$ meaning a point by point multiplication. This newer suit is ruled by v_i and u_i We call it w_i defined by : $v_i * u_i$

In our example, it gives :

$$\begin{cases} w_0 = v_0.u_0 = 324 \\ w_1 = v_1.u_1 = 0 \\ w_2 = v_2.u_2 = 319 \\ w_3 = v_3.u_3 = 980 \\ w_4 = v_5.u_4 = 1292 \end{cases}. \tag{8.4}$$

We obtain the suit w=*324.0.319.980.1292*

8.1.5 Encryption

At the end we use the Encryption into differents numeric bases to hide the crypting process.

The Base indexes are defined by the key ξ
The list to encrypt is defined by w
The Encryption process will be caled Ξ
Defined by :

$$\Xi_i = (w_i)_{\xi_i} \tag{8.5}$$

$$\begin{cases} \Xi_0 = (w_0)_{\xi_0} = (324)_{14} = 192 \\ \Xi_1 = (w_1)_{\xi_1} = (0)_{10} = 0 \\ \Xi_2 = (w_2)_{\xi_2} = (319)_{11} = 270 \\ \Xi_3 = (w_3)_{\xi_3} = (980)_{10} = 980 \\ \Xi_4 = (w_4)_{\xi_4} = (1292)_{14} = 684 \end{cases}. \tag{8.6}$$

The Encrypted suit is Ξ = *192.0.270.980.684*

Its associate key is ξ = *14.10.11.10.14*

8.2 Decrypting Protocol

8.2.1 Initialisation

In this demonstration, we will use a Encrypted list using the Raptor cryptographic algorithm. The terms list is given by :

!018kh"05a3c#8064$12vj%2gai&0605a(67500)0ba30*277a4+25376,2a5db-5813.36u7!146367"27706#1j68c

The associated key is given as a public key :

2116103428141013

We consider in a first time differents type of caracters set used in the crypting and Encrypting processes.

§= [!,",#,$,%,&,(,),*,+,-,.]

Using this informations, we could get a first Terms list to treat called Ξ.

018kh.05a3c.8064.12vj.2gai.0605a.67500.0ba30.277a4.25376.2a5db.5813.36u7.146367.27706.1j68c

A list with length 16 is highlighting We will use the Set X = [a-z] $\cup$ [0-9]

With χ the length of the Terms list.

Here χ = 16, we could observ than length of key $\rho \quad | \quad \rho = \chi$.

Ξ_i will represent the respectives terms of the list.

We start the decrypting process by exctracting the key's Bases index from the c_n number suit contained in key. with c_i, $\forall$ i $\in$ [0,ρ], $c_i \leq 9$

We obtain : ξ = *21.16.10.34.28.14.10.13*

8.2.2 Successive Base Transpositions - Step 1

Highlighted ξ_j , Bases index are consistent with the Terms of the suit Ξ

Thereby, with the Correspondance between ξ_0 and Ξ_0 , we obtain the following chained system resolution.

8.2.3 Ξ_0 = 018kh, ξ_0 = 21

By drawing up the 21 Base Table, we find :

$$\begin{cases} 0 = 0 \\ 1 = 1 \\ 8 = 8 \\ k = 20 \\ h = 17 \end{cases} . \tag{8.7}$$

Or by performing a Base transposition since the 21 Base Table, we obtain :

$$(018kh)_{21} = (0.21^4 + 1.21^3 + 8.21^2 + 20.21 + 17)_{10} = 13226 \tag{8.8}$$

8.2.4 Ξ_1 = 05a3c, ξ_1 = 16

By drawing up the 16 Base Table, we find :

$$\begin{cases} 0 = 0 \\ 5 = 5 \\ a = 10 \\ 3 = 3 \\ c = 12 \end{cases} . \tag{8.9}$$

Or by performing a Base transposition since the 16 Base Table, we obtain :

$$(05a3c)_{16} = (5.16^3 + 10.16^2 + 3.16 + 12)_{10} = 23100 \tag{8.10}$$

8.2.5 Ξ_2 = 8064, ξ_2 = 10

The specified base index ξ_2 = 10, so any conversion is superfluous.

8.2.6 Ξ_3 = 12vj, ξ_3 = 34

By drawing up the 34 Base Table, we find :

$$\begin{cases} 1 = 1 \\ 2 = 2 \\ v = 31 \\ j = 19 \end{cases} . \tag{8.11}$$

Or by performing a Base transposition since the 34 Base Table, we obtain :

$$(12vj)_{34} = (1.34^3 + 2.34^2 + 31.34 + 19)_{10} = 42689 \tag{8.12}$$

8.2.7 Ξ_4 = 2gai, ξ_4 = 28

By drawing up the 28 Base Table, we find :

$$\begin{cases} 2 = 2 \\ g = 16 \\ a = 10 \\ i = 18 \end{cases}. \tag{8.13}$$

Or by performing a Base transposition since the 28 Base Table, we obtain :

$$(2gai)_{28} = (2.28^3 + 16.28^2 + 10.28 + 18)_{10} = 56746 \tag{8.14}$$

8.2.8 Ξ_5 = 0605a, ξ_5 = 14

By drawing up the 14 Base Table, we find :

$$\begin{cases} 0 = 0 \\ 6 = 6 \\ 5 = 5 \\ a = 10 \end{cases}. \tag{8.15}$$

Or by performing a Base transposition since the 14 Base Table, we obtain :

$$(0605a)_{14} = (6.14^3 + 5.14 + 10)_{10} = 16544 \tag{8.16}$$

8.2.9 Ξ_6 = 67500, ξ_6 = 10

The specified base index ξ_6 = 10, so any conversion is superfluous.

8.2.10 Ξ_7 = 0ba30, ξ_7 = 13

By drawing up the 13 Base Table, we find :

$$\begin{cases} b = 11 \\ a = 10 \\ 3 = 3 \\ 0 = 0 \end{cases}. \tag{8.17}$$

Or by performing a Base transposition since the 13 Base Table, we obtain :

$$(0ba30)_{13} = (11.13^3 + 10.13^2 + 3.13 + 13)_{10} = 25886 \tag{8.18}$$

The Base transposition done, we could reverse the key to obtain the rest of the list.

8.2.11 Key build

We can use the following definition :

ρ is the length of the key ξ since Initialisation Section.

We go to compare the ρ length of ξ with χ the length of Ξ.We have χ=2.ρ

We will use the following terms :

- $\tilde{\xi}$: the mirror of ξ
- $\tilde{\xi}_{/n}$: the mirror of ξ bereft of ξ_n
- $\mathring{\xi}$: the rebuilded key

- $\frown$: the concatenation operator

To rebuild the missing half key, we go to reverse ξ with the following syntax

$$\begin{cases} \mathring{\xi} = \xi \frown \tilde{\xi} & if \quad \chi \quad mod \quad 2 = 0 \\ \mathring{\xi} = \xi \frown \tilde{\xi}_{/n} & if \quad \chi \quad mod \quad 2 = 1 \end{cases} . \tag{8.19}$$

8.2.12 Successive Base Transpositions - Step 2

Once the full key rebuilded from ξ, we could transpose again the rest of the list as step 1.

8.2.13 Ξ_8 = 277a4, ξ_8 = 13

By drawing up the 13 Base Table, we find :

$$\begin{cases} 2 = 2 \\ 4 = 4 \\ 7 = 7 \\ a = 10 \end{cases} . \tag{8.20}$$

Or by performing a Base transposition since the 13 Base Table, we obtain :

$$(277a4)_{13} = (2.134 + 7.13^3 + 7.13^2 + 10.13 + 4)_{10} = 73818 \tag{8.21}$$

8.2.14 Ξ_9 = 25376, ξ_9 = 10

The specified base index ξ_9 = 10, so any conversion is superfluous.

8.2.15 Ξ_{10} = 2a5db, ξ_{10} = 14

By drawing up the 14 Base Table, we find :

$$\begin{cases} 2 = 2 \\ 5 = 5 \\ a = 10 \\ b = 11 \\ d = 13 \end{cases} . \tag{8.22}$$

Or by performing a Base transposition since the 14 Base Table, we obtain :

$$(2a5db)_{14} = (2.144 + 10.14^3 + 5.14^2 + 13.14 + 11)_{10} = 105445 \tag{8.23}$$

8.2.16 Ξ_{11} = 5813, ξ_{11} = 28

By drawing up the 28 Base Table, we find :

$$\begin{cases} 1 = 1 \\ 3 = 3 \\ 5 = 5 \\ 8 = 8 \end{cases} . \tag{8.24}$$

Or by performing a Base transposition since the 28 Base Table, we obtain :

$$(5813)_{28} = (5.28^3 + 8.28^2 + 1.28 + 3)_{10} = 116063 \tag{8.25}$$

8.2.17 Ξ_{12} = 36u7, ξ_{12} = 34

By drawing up the 34 Base Table, we find :

$$\begin{cases} 3 = 3 \\ 6 = 6 \\ 7 = 7 \\ u = 30 \end{cases} . \tag{8.26}$$

Or by performing a Base transposition since the 34 Base Table, we obtain :

$$(36u7)_{34} = (3.34^3 + 6.34^2 + 30.34 + 7)_{10} = 125875 \tag{8.27}$$

8.2.18 Ξ_{13} = 146367, ξ_{13} = 10

The specified base index ξ_{13} = 10, so any conversion is superfluous.

8.2.19 Ξ_{14} = 27706, ξ_{14} = 16

Or by performing a Base transposition since the 16 Base Table, we obtain :

$$(27706)_{16} = (2.164 + 7.16^3 + 7.16^2 + 6)_{10} = 161542 \tag{8.28}$$

8.2.20 Ξ_{15} = 1j68c, ξ_{15} = 21

By drawing up the 21 Base Table, we find :

$$\begin{cases} 1 = 1 \\ 6 = 6 \\ 8 = 8 \\ c = 12 \\ j = 19 \end{cases} . \tag{8.29}$$

Or by performing a Base transposition since the 21 Base Table, we obtain :

$$(1j68c)_{21} = (1.214 + 19.21^3 + 6.21^2 + 8.21 + 12)_{10} = 373266 \tag{8.30}$$

We finnaly obtain the following numeric suit :

13226.23100.42689.56746.16544.67500.25886.73818.25376.105445.116063.125875.161542.373266

8.2.21 Chain Polynom Resolution

To continue the decrypting process, we know the suit increasing by recurrence. We can resolve the polynom using logic, we call it Ch.

Ch_n = y^2+(y'^2+(y''^2+...+$y^{(n)2}$)).y + c =0

The recursive injection of a polynome is resolvable uniquely using positive real roots.

With this definition, we will not keep cases with $\triangle \leq 0$

In the last section of the demonstration, we will use the Chain Polynoms resolution algorithm defined by :

- Solve $y^2 + b.y - \Xi_i = 0$
- $x = (root > 0) - b$
- $b = root$
- Add x to the solved list R.

We gonna initialize the procedure with :

- $y^2 = \Xi_0 \iff y = \sqrt{13226} = 115$

  $R_0 = 115$
- $y^2 - 115.y - 23100 = 0$

  $x = 220 - 115 = 105$

  $R_1 = 105$
- $y^2 - 220.y - 8064 = 0$

  $R_2 = 252 - 220 = 32$
- $y^2 - 252.y - 42688 = 0$

  $R_3 = 368 - 252 = 116$
- $y^2 - 368.y - 56745 = 0$

  $R_4 = 485 - 368 = 117$
- $y^2 - 485.y - 16544 = 0$

  $R_5 = 517 - 485 = 32$
- $y^2 - 517.y - 67500 = 0$

  $R_6 = 625 - 517 = 108$
- $y^2 - 625.y - 25896 = 0$

  $R_7 = 664 - 625 = 39$
- $y^2 - 664.y - 73817 = 0$

  $R_8 = 761 - 664 = 97$
- $y^2 - 761.y - 25376 = 0$

  $R_9 = 793 - 761 = 32$
- $y^2 - 793.y - 105444 = 0$

  $R_{10} = 909 - 793 = 116$
- $y^2 - 909.y - 116622 = 0$

  $R_{11} = 1023 - 909 = 114$
- $y^2 - 1023.y - 125874 = 0$

  $R_{12} = 1134 - 1023 = 111$
- $y^2 - 1134.y - 146367 = 0$

  $R_{13} = 1251 - 1134 = 117$
- $y^2 - 1251.y - 161542 = 0$

  $R_{14} = 1369 - 1251 = 118$
- $y^2 - 1369.y - 373266 = 0$

  $R_{15} = 1602 - 1369 = 118$

8.2.22 Conclusion

we can conclude using a simple ASCII table and get letters from the obtained numeric suit.

R={115,105,32,116,117,32,108,39,97,92,116,114,11,117,118,233}

$ASCII_R$={s,i, ,t,u, ,l,',a, ,t,r,o,u,v,é }

We can get the final decrypted string : "si tu l'a trouvé"

CHAPTER

NINE

INDICES AND TABLES

- genindex
- modindex
- search