Le Code Source

DU

```python
from tkinter import *

from tkinter import ttk

from tkinter import filedialog

from tkinter import messagebox

from base64 import b64encode, b64decode

import hashlib

from Crypto.Cipher import AES

import os

from Crypto.Random import get_random_bytes

import random

import pathlib

import sqlite3

from math import gcd

from ttkthemes import ThemedTk

import time

import ctypes

import hashlib

ctypes.windll.shcore.SetProcessDpiAwareness(1)

class database_manager:

    def __init__(self):

        self.con = sqlite3.connect('RSA_CRYPTO.db')

        self.cur = self.con.cursor()

        self.cur.execute("CREATE TABLE IF NOT EXISTS connexion(utilisateur VARCHAR PRIMARY KEY, mot_de_passe VARCHAR)")

        self.cur.execute("CREATE TABLE IF NOT EXISTS donnees_cryptage(utilisateur VARCHAR,date DATE,fichier,dictionnaire de cles,FOREIGN KEY (utilisateur) REFERENCES connexion(utilisateur))")

        self.con.commit()

    def partie_objet_tableau(self, nom_tableau, user = '',colonne = 0):

        if nom_tableau == 'connexion':

            self.cur.execute(f'SELECT * FROM {nom_tableau}')

        elif nom_tableau == 'donnees_cryptage':

            self.cur.execute(f'SELECT * FROM {nom_tableau} WHERE utilisateur = (?)',(user,))

        donnees= self.cur.fetchall()

        self.con.commit()

        colonne_tuple = ()

        for donnee in donnees:

            colonne_tuple += (donnee[colonne],)

        return (colonne_tuple, donnees)

    def trouver_objet(self, nom_tableau, objet, colonne):

        self.cur.execute(f'SELECT * FROM {nom_tableau} WHERE "{colonne}" = (?)',(objet,))

        donnees = self.cur.fetchall()

        self.cur.commit()

        if donnees!=[]:
```

```
                return True

        return False


    def ajouter_objet(self, nom_tableau, objet):

        if nom_tableau == 'connexion':

            self.cur.execute(f'INSERT INTO {nom_tableau} VALUES(?,?)',objet)

            self.con.commit()

        elif nom_tableau == 'donnees_cryptage':

            self.cur.execute(f'INSERT INTO {nom_tableau} VALUES(?,?,?,?)',objet)

            self.con.commit()

    def supprimer_objet(self, nom_tableau, colonne, objet):

        self.cur.execute(f'DELETE FROM {nom_tableau} WHERE {colonne} = (?)',(objet,))

        self.con.commit()


class MainWindow(ThemedTk):

    def __init__(self,bd):

        ThemedTk.__init__(self, theme="arc")

        self.bd = bd

        self.mainWidgets()

    def mainWidgets(self):

        self.title('RSA Crypto')

        self.iconbitmap(self,'RSA_CRYPTO_inicon.ico')

        x = self.winfo_screenwidth()//2 - 400//2

        y = self.winfo_screenheight()//2 - 500//2

        self.geometry(f'400x500+{x}+{y}')

        self.resizable(0,0)

        self.configure(bg='#FFFFFF')

        self.fenetre = fenetre1(self, self.bd)

class fenetre1(Frame):

    def __init__(self, parent, bd):

        Frame.__init__(self, parent, bg = '#FFFFFF')

        self.place(relx=0.5, rely=0.5, anchor=CENTER)

        self.parent = parent

        self.mainWidgets()

    def mainWidgets(self):

        #insertion de l'image
d'entree---------------------------------------------------------------------------------------------
---------

        self.img = PhotoImage(file = r"RSA_CRYPTO_home.png")

        self.photo = Label(self, image=self.img, bg  = '#FFFFFF')

        #Partie de nom
d'utilisateur-----------------------------------------------------------------------------------------
---------------

        self.utilisateur_lb = Label(self,text = 'Utilisateur',bg = '#FFFFFF',font = ('arial',
10, 'bold'))

        self.utilisateur = StringVar()
```

```
162
163         self.utilisateur_en = ttk.Combobox(self, values =
    bd.partie_objet_tableau('connexion')[0], textvariable = self.utilisateur)
164
165         #Partie de mot de
    passe-------------------------------------------------------------------------------------------
    -------------
166
167         self.mot_de_passe = StringVar()
168
169         self.mot_de_passe_lb = Label(self,text = 'Mot de passe',bg = '#FFFFFF',font = ('arial',
    10, 'bold'))
170
171         self.mot_de_passe_en = ttk.Entry(self, show = '*', textvariable=self.mot_de_passe)
172
173         #partie bouton de
    connection---------------------------------------------------------------------------------------
    -------------
174
175         self.se_connecter_btn = ttk.Button(self, text = 'Se connecter',command  =
    self.se_connecter)
176
177         self.inscrire_btn = ttk.Button(self, text = "S'inscrire",command  = self.inscrire)
178
179         self.supprimer_btn = ttk.Button(self, text = "Supprimer compte",command  =
    self.supprimer)
180
181         #Creation des
    extentions----------------------------------------------------------------------------------------
    -----------------
182
183         self.photo.grid(row = 0, column = 1,columnspan = 2, pady = 5)
184
185         self.utilisateur_lb.grid(row = 1, column = 1,columnspan = 2, pady = 5)
186
187         self.utilisateur_en.grid(row = 2, column = 1,columnspan = 2, pady = 5)
188
189         self.mot_de_passe_lb.grid(row = 3, column = 1,columnspan = 2, pady = 5)
190
191         self.mot_de_passe_en.grid(row = 4, column = 1,columnspan = 2, pady = 5)
192
193         self.se_connecter_btn.grid(row = 5, column = 1,columnspan = 2, pady = 5)
194
195         self.inscrire_btn.grid(row = 6, column = 1,columnspan = 2, pady = 5)
196
197         self.supprimer_btn.grid(row = 7, column = 1,columnspan = 2, pady = 5)
198
199
    #-------------------------------------------------------------------------------------------------
    -------------------------------
200
201
    #-------------------------------------------------------------------------------------------------
    -------------------------------
202
203
    ###################################################################################################
    ###################################
204     #-----------------------------------------------------Fonction
    Predefinie-----------------------------------------------------#
205
    ###################################################################################################
    ###################################
206
207     def clear_frame(self):
208
209         for widgets in self.winfo_children():
210
211             widgets.destroy()
212
213     #se connecter au compte
    choisie-----------------------------------------------------------------------------------------
    --------
214
215     def se_connecter(self):
216
217         if self.utilisateur_en.get() == '' or self.mot_de_passe_en.get() == '' :
218
219             messagebox.showerror('Alert', 'Champs vides !!')
220
221         else:
222
223             for donnee in bd.partie_objet_tableau('connexion')[1]:
224
```

```
                if self.utilisateur_en.get() == donnee[0] and
hashlib.md5(self.mot_de_passe_en.get().encode()).hexdigest() == donnee[1]:

                    user = self.utilisateur.get()

                    self.utilisateur.set('')

                    self.mot_de_passe.set('')

                    test = messagebox.askokcancel('Information', 'Voulez vous continuer?')

                    if test == True:

                        self.clear_frame()

                        self.fenetre_cryptage = fenetre3(self.parent, bd, user,
self.mot_de_passe.get())

                        self.fenetre_cryptage.mainloop()

                        break

                    else:
                        break

            else:
                messagebox.showerror('Alert', "Nom d'utilisateur ou mot de passe est incorrect")


#-------------------------------------------------------------------------------------------------
------------------------------


    #se connecter au compte
choisie-------------------------------------------------------------------------------------------
--------

    def inscrire(self):

        self.clear_frame()

        self.fenetre_inscription = fenetre2(self.parent, bd)

        self.fenetre_inscription.mainloop()

    def supprimer(self):

        if self.utilisateur_en.get() == '' or self.mot_de_passe_en.get() == '' :

            messagebox.showerror('Alert', 'Champs vides !!')

        else:

            for donnee in bd.partie_objet_tableau('connexion')[1]:

                if self.utilisateur_en.get() == donnee[0] and
hashlib.md5(self.mot_de_passe_en.get().encode()).hexdigest() == donnee[1]:

                    test = messagebox.askokcancel('Alert', 'La suppression est definitive
\nVoulez vous continuer?')

                    if test == True:

                        bd.supprimer_objet('connexion','utilisateur' , self.utilisateur_en.get())

                        bd.supprimer_objet('donnees_cryptage','utilisateur' ,
self.utilisateur_en.get())

                        self.clear_frame()

                        self.fenetre_connexion = fenetre1(self.parent, bd)

                        self.fenetre_connexion.mainloop()

                    break

            else:
                messagebox.showerror('Alert', "Nom d'utilisateur ou mot de passe est incorrect")


#-------------------------------------------------------------------------------------------------
------------------------------


```

```
298 class fenetre2(Frame):
299
300     def __init__(self, parent, bd):
301
302         Frame.__init__(self, parent, bg = '#FFFFFF')
303
304         self.place(relx=0.5, rely=0.5, anchor=CENTER)
305
306         self.parent = parent
307
308         self.mainWidgets()
309
310     def mainWidgets(self):
311
312         #Partie de nom
    d'utilisateur----------------------------------------------------------------------------------
    ---------------
313
314         self.utilisateur_ins_lb = Label(self,text = 'Utilisateur',bg = '#FFFFFF',font =
    ('arial', 10, 'bold'))
315
316         self.utilisateur_ins = StringVar()
317
318         self.utilisateur_ins_en = ttk.Entry(self, textvariable = self.utilisateur_ins)
319
320         #Partie de mot de
    passe------------------------------------------------------------------------------------
    -------------
321
322         self.mot_de_passe_ins = StringVar()
323
324         self.mot_de_passe_ins_lb = Label(self,text = 'Mot de passe',bg = '#FFFFFF',font =
    ('arial', 10, 'bold'))
325
326         self.mot_de_passe_ins_en = ttk.Entry(self, show = '*',
    textvariable=self.mot_de_passe_ins)
327
328         self.conf_mot_de_passe_ins = StringVar()
329
330         self.conf_mot_de_passe_lb = Label(self,text = 'Confirmer mot de passe',bg =
    '#FFFFFF',font = ('arial', 10, 'bold'))
331
332         self.conf_mot_de_passe_en = ttk.Entry(self, show = '*',
    textvariable=self.conf_mot_de_passe_ins)
333
334         #partie bouton de
    connection-------------------------------------------------------------------------------
    -------------
335
336         self.valider_inscription_btn = ttk.Button(self, text = 'Valider',command  = self.valider)
337
338         self.retour_btn = ttk.Button(self, text = "Retour",command  = self.retour)
339
340         #Creation des
    extentions-------------------------------------------------------------------------------
    ------------------
341         self.utilisateur_ins_lb.grid(row = 1, column = 1,columnspan = 2, pady = 5)
342
343         self.utilisateur_ins_en.grid(row = 2, column = 1,columnspan = 2, pady = 5)
344
345         self.mot_de_passe_ins_lb.grid(row = 3, column = 1,columnspan = 2, pady = 5)
346
347         self.mot_de_passe_ins_en.grid(row = 4, column = 1,columnspan = 2, pady = 5)
348
349         self.conf_mot_de_passe_lb.grid(row = 5, column = 1,columnspan = 2, pady = 5)
350
351         self.conf_mot_de_passe_en.grid(row = 6, column = 1,columnspan = 2, pady = 5)
352
353         self.valider_inscription_btn.grid(row = 7, column = 1,columnspan = 2, pady = 5)
354
355         self.retour_btn.grid(row = 8, column = 1,columnspan = 2, pady = 5)
356
357
    #----------------------------------------------------------------------------------------
    ------------------------------
358
359
    ##########################################################################################
    ################################
360     #-------------------------------------------------------Fonction
    Predefinie-------------------------------------------------------#
361
    ##########################################################################################
    ################################
```

```
362
363     def clear_frame(self):
364
365         for widgets in self.winfo_children():
366
367             widgets.destroy()
368
369     def valider(self):
370
371         if self.utilisateur_ins_en.get()!='' and self.mot_de_passe_ins_en.get()!='' and
    self.conf_mot_de_passe_en.get()!='':
372
373             colonne_tuple = bd.partie_objet_tableau('connexion')[0]
374
375             for user in colonne_tuple:
376
377                 if self.utilisateur_ins_en.get() == user:
378
379                     self.utilisateur_ins.set('')
380
381                     self.mot_de_passe_ins.set('')
382
383                     self.conf_mot_de_passe_ins.set('')
384
385                     return messagebox.showerror('Alert', 'Ce utilisateur existe deja')
386
387             if self.mot_de_passe_ins_en.get() != self.conf_mot_de_passe_en.get():
388
389                 self.mot_de_passe_ins.set('')
390
391                 self.conf_mot_de_passe_ins.set('')
392
393                 messagebox.showerror('Alert', 'Les mots de passes sont differents')
394
395             elif len(self.mot_de_passe_ins_en.get()) < 8:
396
397                 self.mot_de_passe_ins.set('')
398
399                 self.conf_mot_de_passe_ins.set('')
400
401                 messagebox.showerror('Alert', 'Mot de passe est faible \n veuiller depasser 8
    caracteres')
402
403             else :
404
405                 test = messagebox.askokcancel('Information', 'Voulez vous continuer?')
406
407                 if test == True:
408
409                     mot_de_passe_securiser =
    hashlib.md5(self.mot_de_passe_ins_en.get().encode()).hexdigest()
410
411                     bd.ajouter_objet('connexion',
    (self.utilisateur_ins_en.get(),mot_de_passe_securiser))
412
413                     self.clear_frame()
414
415                     self.fenetre_de_connection = fenetre1(self.parent, bd)
416
417                     self.fenetre_de_connection.mainloop()
418
419
420         else:
421             messagebox.showerror('Alert', 'Champs vides !!')
422
423
424
425     def retour(self):
426
427         self.clear_frame()
428
429         self.fenetre_de_connection = fenetre1(self.parent, bd)
430
431         self.fenetre_de_connection.mainloop()
432
433 #-------------------------------------------------------------------------------------------------
    ------------------------------
434
435
436
437 class fenetre3(Frame):
438
439     def __init__(self, parent, bd, user, password):
440
```

```
441         Frame.__init__(self, parent, bg = '#FFFFFF')
442 
443         self.place(relx=0.5, rely=0.5, anchor=CENTER)
444 
445         self.parent = parent
446 
447         self.user = user
448 
449         self.password = password
450 
451         self.chemainStr = ''
452 
453         self.mainWidgets()
454 
455     def mainWidgets(self):
456 
457         #insertion du fichier de cryptage
    ----------------------------------------------------------------------------------------------
458 
459         self.inserer_lb = Label(self,text = 'Inserer Fichier',bg = '#FFFFFF',font = ('arial',
    10, 'bold'))
460 
461         self.chemain = StringVar()
462 
463         self.inserer_en = ttk.Entry(self, textvariable=self.chemain)
464 
465         self.ouvrir_btn = ttk.Button(self, text = 'Ouvrir', command = self.fileINopen)
466 
467         #Choix entre cryptage ou decryptage
    ----------------------------------------------------------------------------------------------
468 
469         self.choix = StringVar()
470 
471         s = ttk.Style()
472 
473         s.configure('white.TRadiobutton', background="#FFFFFF")
474 
475         self.choix1 = ttk.Radiobutton(self,text = 'Crypter',value = 'crypter',variable =
    self.choix, style = 'white.TRadiobutton', command = self.click_rbtn)
476 
477         self.choix2 = ttk.Radiobutton(self,text = 'Decrypter',value = 'decrypter',variable =
    self.choix, style = 'white.TRadiobutton', command = self.click_rbtn)
478 
479         #Choix de cle de cryptage
    ----------------------------------------------------------------------------------------------
    -----
480 
481         self.cle = StringVar()
482 
483         valeurs = bd.partie_objet_tableau('donnees cryptage',self.user,2)[0]
484 
485         vals = ()
486 
487         for i in range(len(valeurs)):
488 
489             vals += (str(i+1)+'-'+valeurs[i],)
490 
491         self.cle_privee = ttk.Combobox(self, values = vals, textvariable = self.cle,state
    ='disable')
492 
493         #Traitement du
    self.choix----------------------------------------------------------------------------------------
    ----------------------
494 
495         self.executer_btn = ttk.Button(self, text = 'Executer', command = self.executer,state =
    'disable')
496 
497         #Affichange de l'historique
498 
499         self.historique_btn = ttk.Button(self, text = 'historique', command = self.historique)
500 
501         # Se deconnecter
502 
503         self.se_deconnecter_btn = ttk.Button(self, text = 'Se deconnecter', command =
    self.se_deconnecter)
504 
505         #Creation des
    extentions-----------------------------------------------------------------------------------
    ------------------
506 
507         self.historique_btn.grid(row = 6, column = 0,columnspan = 2, pady = 5)
508 
509         self.se_deconnecter_btn.grid(row = 6, column = 2,columnspan = 2, pady = 5)
510 
```

```
511             self.inserer_lb.grid(row = 0, column = 1,columnspan = 2, pady = 5)
512
513             self.inserer_en.grid(row = 1, column = 1, columnspan = 2, pady = 5)
514
515             self.ouvrir_btn.grid(row = 2, column = 1, columnspan = 2, pady = 5)
516
517             self.choix1.grid(row = 3, column = 1,pady = 20,padx = 5)
518
519             self.choix2.grid(row = 3, column = 2,pady = 20,padx = 5)
520
521             self.cle_privee.grid(row = 4, column = 1, columnspan = 2, pady = 5)
522
523             self.executer_btn.grid(row = 5, column = 1, columnspan = 2, pady = 5)
524
525
    #-------------------------------------------------------------------------------------------------------------------------------
526
527
528
    ##################################################################################################################################
529         #-------------------------------------------------------Fonction Predefinie-------------------------------------------------------#
530     ##################################################################################################################################
531
532         # l'insertion du fichier pour le crypter-------------------------------------------------------------------------------------
533
534         def clear_frame(self):
535
536             for widgets in self.winfo_children():
537
538                 widgets.destroy()
539
540         def fileINopen(self):
541
542                 self.chemainStr = filedialog.askopenfilename()
543
544                 filename = self.chemainStr.split('/')[-1]
545
546                 self.chemain.set(filename)
547
548                 return self.chemainStr
549
550
    #-------------------------------------------------------------------------------------------------------------------------------
551
552         def historique(self):
553
554             self.fenetre_historique = fenetre4(self, self.parent, bd, self.user, self.password)
555
556             self.fenetre_historique.mainloop()
557
558
559     #None-------------------------------------------------------------------------------------------------------------------------
560
561         def executer(self):
562
563             if self.chemain.get() != '':
564
565                 if self.choix.get() == 'crypter':
566
567                     date = time.strftime("%D-%H:%M:%S", time.localtime())
568
569                     def est_premier(num):
570                         if num == 2:
571                             return True
572                         if num < 2 or num % 2 == 0:
573                             return False
574                         for n in range(3, int(num**0.5)+2, 2):
575                             if num % n == 0:
576                                 return False
577                         return True
578
579                     def generateurDesCles():
580                         p = generateurDesNbrPremier()
581                         q = generateurDesNbrPremier()
582                         n = p*q
```

```
583                     phi = (p-1) * (q-1)
584                     e = random.randint(1, phi)
585                     g = gcd(e,phi)
586                     while g != 1:
587                         e = random.randint(1, phi)
588                         g = gcd(e, phi)
589                     d = egcd(e, phi)[1]
590                     d = d % phi
591                     if(d < 0):
592                         d += phi
593 
594                     return ((e,n),(d,n))
595 
596                 def generateurDesNbrPremier(keysize=1000):
597                     while True:
598                         ranPremier = random.randint(50,keysize)
599                         if est_premier(ranPremier):
600                             return ranPremier
601 
602 
603                 def egcd(a, b):
604                     s = 0; old_s = 1
605                     t = 1; old_t = 0
606                     r = b; old_r = a
607 
608                     while r != 0:
609                         quotient = old_r // r
610                         old_r, r = r, old_r - quotient * r
611                         old_s, s = s, old_s - quotient * s
612                         old_t, t = t, old_t - quotient * t
613 
614                     # return gcd, x, y
615                     return old_r, old_s, old_t
616 
617                 def cryptage(public_key, msg):
618                     key,n = public_key
619                     cipher = ""
620 
621                     for c in msg:
622                         m = ord(c)
623                         cipher += str(pow(m, key, n)) + " "
624                     return cipher
625 
626                 public_key,private_key = generateurDesCles()
627 
628                 try:
629 
630                     fI = open(self.chemainStr,'rb')
631 
632                     msg = fI.read()
633 
634                     ctext = cryptage(public_key, msg.decode("ansi"))
635 
636                     extention = self.chemain.get().split('.')[-1]
637 
638                     data = [(f'*.{extention}', f'*.{extention}')]
639 
640                     chemin = filedialog.asksaveasfilename(confirmoverwrite=False, filetypes =
    data, defaultextension = data)
641 
642                     fO = open(chemin,'wb')
643                     fO.write(ctext.encode("ansi"))
644                     fO.close()
645                     fI.close()
646 
647                     private_key = bdcrypto(self.password, str(private_key)).encrypt()
648 
649                     bd.ajouter_objet('donnees_cryptage',(self.user, date,
    self.chemain.get(),str(private_key)))
650 
651                     messagebox.showinfo('info', 'Le cryptage est termine <^_^>' )
652 
653                     self.clear_frame()
654 
655                     self.fenetre_cryptage = fenetre3(self.parent, bd, self.user, self.password)
656 
657                     self.fenetre_cryptage.mainloop()
658 
659                 except Exception as e:
660 
661                     messagebox.showerror('info', e)
662 
663 
664             elif self.choix.get() == 'decrypter':
```

```
                if self.cle_privee.get()  == '':

                    messagebox.showerror('Alert', 'Champs vides!!')

                else:

                    test = False

                    valeurs = bd.partie_objet_tableau('donnees_cryptage',self.user,2)[0]

                    for i in range(len(valeurs)):

                        if (str(i+1)+'-'+valeurs[i]) == self.cle_privee.get():

                            test = True

                            break

                    if test == True:

                        index = int(self.cle_privee.get()[0])

                        val = eval(bd.partie_objet_tableau('donnees_cryptage',self.user,3)[0][index-1])

                        private_key = eval(bdcrypto(self.password, '',val).decrypt())

                        def decryptage(private_key, cipher):

                            key,n = private_key
                            msg = ""
                            parts = cipher.split()
                            for part in parts:
                                if part:
                                    c = int(part)
                                    msg += chr(pow(c, key, n))

                            return msg

                        try:
                            fI = open(self.chemainStr,'rb')
                            msg = fI.read()

                            text = decryptage(private_key, msg.decode("ansi"))

                            extention = self.chemain.get().split('.')[-1]

                            data = [(f'*.{extention}', f'*.{extention}')]

                            chemin = filedialog.asksaveasfilename(confirmoverwrite=False, filetypes = data, defaultextension = data)

                            fO = open(chemin,'wb')
                            text = text.encode("ansi")
                            fO.write(text)
                            fO.close()
                            fI.close()

                            messagebox.showinfo('info', 'Le decryptage est termine <^_^>' )

                            self.clear_frame()

                            self.fenetre_cryptage = fenetre3(self.parent, bd, self.user, self.password)

                            self.fenetre_cryptage.mainloop()

                        except Exception as e:

                            messagebox.showerror('info', e)

                    else:

                        messagebox.showerror('Alert', 'Cle privee est imcompatible!')


        else:

            messagebox.showerror('Alert', 'Champs vides!!')
```

```
    #-------------------------------------------------------------------------------------------------------------------------------
746
747
748     #Activation et desactivation de la partie de la selection des cles-----------------------------------------------------------
749
750     def click_rbtn(self):
751
752         test = self.choix.get()
753
754         self.executer_btn.config(state = 'normale')
755
756         if test =='crypter':
757
758             self.cle_privee.config(state = 'disable')
759
760         elif test =='decrypter':
761
762             self.cle_privee.config(state = 'normal')
763
764
    #-------------------------------------------------------------------------------------------------------------------------------
765
766
767
768
    #-------------------------------------------------------------------------------------------------------------------------------
769
770
771     #commande de deconnection et l'affiche de la partie connection-----------------------------------------------------------------
772
773     def se_deconnecter(self):
774
775         self.clear_frame()
776
777         self.fenetre_de_connection = fenetre1(self.parent, bd)
778
779         self.fenetre_de_connection.mainloop()
780
781
    #-------------------------------------------------------------------------------------------------------------------------------
782
783 class fenetre4(Toplevel):
784
785     def __init__(self,manager, parent, bd, user, password):
786
787         Toplevel.__init__(self, parent, bg = '#FFFFFF')
788
789         self.grab_set()
790
791         self.parent = parent
792
793         self.manager = manager
794
795         self.password = password
796
797         self.user = user
798
799         self.mainWidgets()
800
801     def mainWidgets(self):
802
803         self.title('Historique de cryptage')
804
805         x = self.winfo_screenwidth()//2 - 900//2
806
807         y = self.winfo_screenheight()//2 - 400//2
808
809         self.geometry(f'+{x}+{y}')
810
811         self.resizable(0,0)
812
813         # Titre du self.tableau-------------------------------------------------------------------------------------------------------
814
815         self.historique_lb = Label(self,bg = "white", text = 'Historique')
816
817         self.historique_lb.grid(row = 0, column = 4,padx = (0,10), pady=(5, 0))
```

```
818
819         #construction du self.tableau
    d'historique----------------------------------------------------------------------------------
    ------
820
821         self.tableau = ttk.Treeview(self, columns=('date', 'fichier'))
822
823         self.tableau.heading('date', text='Date')
824
825         self.tableau.heading('fichier', text='Fichier')
826
827         self.tableau['show'] = 'headings' # sans ceci, il y avait une colonne vide à gauche qui
    a pour rôle d'afficher le paramètre "text" qui peut être spécifié lors du insert
828
829         self.sbar = ttk.Scrollbar(self, orient="vertical", command = self.tableau.yview)
830
831         self.tableau.grid(row = 1, column = 1,columnspan = 6,padx = (5,0), pady=(5,
    0),sticky="news",rowspan = 5)
832
833         self.sbar.grid(row = 1, column = 7, rowspan = 5, pady=(5, 0),sticky='ns')
834
835         self.tableau.configure(yscrollcommand=self.sbar.set)
836
837         # bouton pour fermer la
    self.fenetre historique-----------------------------------------------------------------------
    ----------------------
838
839         self.fermer_btn = ttk.Button(self, text = 'Fermer', command = self.destroy)
840
841         self.supprimer_btn = ttk.Button(self, text = 'Supprimer', command =
    self.supprimer_element)
842
843         self.fermer_btn.grid(row = 6, column = 1, pady=5)
844
845         self.supprimer_btn.grid(row = 6, column = 6, pady=5)
846
847         donnees = bd.partie_objet_tableau('donnees_cryptage',self.user)[1]
848
849         for donnee in donnees:
850
851             self.tableau.insert('', 'end', iid=donnee[1], values=(donnee[1], donnee[2]))
852
853
854     def supprimer_element (self):
855
856         selected_items = self.tableau.selection()
857
858         if selected_items !=():
859
860             test = messagebox.askokcancel('Alert', 'La suppression est definitive \nVoulez vous
    continuer?')
861
862             if test == True:
863
864                 for element_selectionnee in selected_items:
865
866                     bd.supprimer_objet('donnees_cryptage', 'date', element_selectionnee)
867
868                     self.tableau.delete(element_selectionnee)
869
870                     self.manager.destroy()
871
872                     self.fenetre3 = fenetre3(self.parent, bd, self.user, self.password)
873
874         else:
875
876             messagebox.showerror('Erreur', 'Vous devez selectionner un element')
877
878 class bdcrypto():
879
880     def __init__(self, password,texte = '',enc_dict = {}):
881
882         self.password = password
883
884         self.texte = texte
885
886         self.enc_dict = enc_dict
887
888     def encrypt(self):
889
890         salt = get_random_bytes(AES.block_size)
891
892         private_key = hashlib.scrypt(self.password.encode(), salt=salt, n=2**14, r=8, p=1,
    dklen=32)
```

```
893
894             cipher_config = AES.new(private_key, AES.MODE_GCM)
895
896             cipher_text, tag = cipher_config.encrypt_and_digest(bytes(self.texte, 'utf-8'))
897
898             return {
899                 'cipher_text': b64encode(cipher_text).decode('utf-8'),
900                 'salt': b64encode(salt).decode('utf-8'),
901                 'nonce': b64encode(cipher_config.nonce).decode('utf-8'),
902                 'tag': b64encode(tag).decode('utf-8')
903             }
904
905
906         def decrypt(self):
907
908             # decode the dictionary entries from base64
909             salt = b64decode(self.enc_dict['salt'])
910             cipher_text = b64decode(self.enc_dict['cipher_text'])
911             nonce = b64decode(self.enc_dict['nonce'])
912             tag = b64decode(self.enc_dict['tag'])
913
914             # generate the private key from the password and salt
915             private_key = hashlib.scrypt(self.password.encode(), salt=salt, n=2**14, r=8, p=1,
    dklen=32)
916
917             # create the cipher config
918             cipher = AES.new(private_key, AES.MODE_GCM, nonce=nonce)
919
920             # decrypt the cipher text
921             decrypted = cipher.decrypt_and_verify(cipher_text, tag)
922
923             return decrypted.decode()
924
925
926
    #-------------------------------------------------------------------------------------------------
    ------------------------------
927
928
    #-------------------------------------------------------------------------------------------------
    ------------------------------
929
930
931
932
933     if __name__=="__main__":
934
935         bd = database_manager()
936
937         w = MainWindow(bd)
938
939         w.mainloop()
940
```