```
import os
import sys
import hashlib
import tkinter as tk
from tkinter import filedialog
from tkinter import messagebox
from Cryptodome.Cipher import AES
# import threading


class EncryptionTool:
    """ "EncryptionTool" class from "github.com/nsk89" for file encryption.
    (Has been modified a bit.) """
    def __init__(self, user_file, user_key, user_salt):
        # get the path to input file
        self.user_file = user_file

        self.input_file_size = os.path.getsize(self.user_file)
        self.chunk_size = 1024
        self.total_chunks = (self.input_file_size // self.chunk_size) + 1

        # convert the key and salt to bytes
        self.user_key = bytes(user_key, "utf-8")
        self.user_salt = bytes(user_key[::-1], "utf-8")

        # get the file extension
        self.file_extension = self.user_file.split(".")[-1]

        # hash type for hashing key and salt
        self.hash_type = "SHA256"

        # encrypted file name
        self.encrypt_output_file = ".".join(self.user_file.split(".")[:-1]) \
            + "." + self.file_extension + ".kryp"

        # decrypted file name
        self.decrypt_output_file = self.user_file[:-5].split(".")
        self.decrypt_output_file = ".".join(self.decrypt_output_file[:-1]) \
            + "__dekrypted__." + self.decrypt_output_file[-1]

        # dictionary to store hashed key and salt
        self.hashed_key_salt = dict()

        # hash key and salt into 16 bit hashes
        self.hash_key_salt()

    def read_in_chunks(self, file_object, chunk_size=1024):
        """Lazy function (generator) to read a file piece by piece.
        Default chunk size: 1k.
        Code Courtesy:
https://stackoverflow.com/questions/519633/lazy-method-for-reading-big-file-in-python
        """
        while True:
            data = file_object.read(chunk_size)
            if not data:
                break
            yield data

    def encrypt(self):
        # create a cipher object
        cipher_object = AES.new(
            self.hashed_key_salt["key"],
            AES.MODE_CFB,
            self.hashed_key_salt["salt"]
        )

        self.abort() # if the output file already exists, remove it first

        input_file = open(self.user_file, "rb")
        output_file = open(self.encrypt_output_file, "ab")
        done_chunks = 0

        for piece in self.read_in_chunks(input_file, self.chunk_size):
            encrypted_content = cipher_object.encrypt(piece)
            output_file.write(encrypted_content)
            done_chunks += 1
            yield (done_chunks / self.total_chunks) * 100

        input_file.close()
        output_file.close()

        # clean up the cipher object
        del cipher_object

    def decrypt(self):
        #  exact same as above function except in reverse
        cipher_object = AES.new(
            self.hashed_key_salt["key"],
            AES.MODE_CFB,
            self.hashed_key_salt["salt"]
        )

        self.abort() # if the output file already exists, remove it first

        input_file = open(self.user_file, "rb")
        output_file = open(self.decrypt_output_file, "xb")
        done_chunks = 0

        for piece in self.read_in_chunks(input_file):
            decrypted_content = cipher_object.decrypt(piece)
            output_file.write(decrypted_content)
            done_chunks += 1
            yield (done_chunks / self.total_chunks) * 100

        input_file.close()
        output_file.close()

        # clean up the cipher object
        del cipher_object

    def abort(self):
        if os.path.isfile(self.encrypt_output_file):
            os.remove(self.encrypt_output_file)
        if os.path.isfile(self.decrypt_output_file):
            os.remove(self.decrypt_output_file)


    def hash_key_salt(self):
        # --- convert key to hash
        #  create a new hash object
        hasher = hashlib.new(self.hash_type)
        hasher.update(self.user_key)

        # turn the output key hash into 32 bytes (256 bits)
        self.hashed_key_salt["key"] = bytes(hasher.hexdigest()[:32], "utf-8")

        # clean up hash object
        del hasher

        # --- convert salt to hash
        #  create a new hash object
        hasher = hashlib.new(self.hash_type)
        hasher.update(self.user_salt)

        # turn the output salt hash into 16 bytes (128 bits)
        self.hashed_key_salt["salt"] = bytes(hasher.hexdigest()[:16], "utf-8")

        # clean up hash object
        del hasher


# class EncryptionThread(threading.Thread):
#     mutual_space = {}
#     threadLock = threading.Lock()

#     def __init__(self, index):
#         threading.Thread.__init__(self)
#         self.threadID = index

#     def run(self):
#         try:
#             pass
#         except Exception as e:
#             print(e)
#             return

#         # Get lock to synchronize threads
#         self.threadLock.acquire()
#         # Append stuff to mutual_space

#         # Free lock to release next thread
#         self.threadLock.release()


class MainWindow:
    """ GUI Wrapper """

    # configure root directory path relative to this file
    THIS_FOLDER_G = ""
    if getattr(sys, "frozen", False):
        # frozen
        THIS_FOLDER_G = os.path.dirname(sys.executable)
    else:
        # unfrozen
        THIS_FOLDER_G = os.path.dirname(os.path.realpath(__file__))
```

```
def __init__(self, root):
    self.root = root
    self._cipher = None
    self._file_url = tk.StringVar()
    self._secret_key = tk.StringVar()
    self._salt = tk.StringVar()
    self._status = tk.StringVar()
    self._status.set("---")

    self.should_cancel = False

    root.title("KrypApp")
    root.configure(bg="#eeeeee")

    try:
        icon_img = tk.Image(
            "photo",
            file=self.THIS_FOLDER_G + "/assets/icon.png"
        )
        root.call(
            "wm",
            "iconphoto",
            root._w,
            icon_img
        )
    except Exception:
        pass

    self.menu_bar = tk.Menu(
        root,
        bg="#eeeeee",
        relief=tk.FLAT
    )
    self.menu_bar.add_command(
        label="How To",
        command=self.show_help_callback
    )
    self.menu_bar.add_command(
        label="Quit!",
        command=root.quit
    )

    root.configure(
        menu=self.menu_bar
```

```
)

self.file_entry_label = tk.Label(
    root,
    text="Enter File Path Or Click SELECT FILE Button",
    bg="#eeeeee",
    anchor=tk.W
)
self.file_entry_label.grid(
    padx=12,
    pady=(8, 0),
    ipadx=0,
    ipady=1,
    row=0,
    column=0,
    columnspan=4,
    sticky=tk.W+tk.E+tk.N+tk.S
)

self.file_entry = tk.Entry(
    root,
    textvariable=self._file_url,
    bg="#fff",
    exportselection=0,
    relief=tk.FLAT
)
self.file_entry.grid(
    padx=15,
    pady=6,
    ipadx=8,
    ipady=8,
    row=1,
    column=0,
    columnspan=4,
    sticky=tk.W+tk.E+tk.N+tk.S
)

self.select_btn = tk.Button(
    root,
    text="SELECT FILE",
    command=self.selectfile_callback,
    width=42,
    bg="#1089ff",
    fg="#ffffff",
    bd=2,
    relief=tk.FLAT
)
self.select_btn.grid(
    padx=15,
    pady=8,
    ipadx=24,
    ipady=6,
    row=2,
    column=0,
    columnspan=4,
    sticky=tk.W+tk.E+tk.N+tk.S
)

self.key_entry_label = tk.Label(
    root,
    text="Enter Secret Key (Remember this for Decryption)",
    bg="#eeeeee",
    anchor=tk.W
)
self.key_entry_label.grid(
    padx=12,
    pady=(8, 0),
    ipadx=0,
    ipady=1,
    row=3,
    column=0,
    columnspan=4,
    sticky=tk.W+tk.E+tk.N+tk.S
)

self.key_entry = tk.Entry(
    root,
    textvariable=self._secret_key,
    bg="#fff",
    exportselection=0,
    relief=tk.FLAT
)
self.key_entry.grid(
    padx=15,
    pady=6,
    ipadx=8,
    ipady=8,
    row=4,
    column=0,
    columnspan=4,
    sticky=tk.W+tk.E+tk.N+tk.S
)

self.encrypt_btn = tk.Button(
    root,
    text="ENCRYPT",
    command=self.encrypt_callback,
    bg="#ed3833",
    fg="#ffffff",
    bd=2,
    relief=tk.FLAT
)
self.encrypt_btn.grid(
    padx=(15, 6),
    pady=8,
    ipadx=24,
    ipady=6,
    row=7,
    column=0,
    columnspan=2,
    sticky=tk.W+tk.E+tk.N+tk.S
)

self.decrypt_btn = tk.Button(
    root,
    text="DECRYPT",
    command=self.decrypt_callback,
    bg="#00bd56",
    fg="#ffffff",
    bd=2,
    relief=tk.FLAT
)
self.decrypt_btn.grid(
    padx=(6, 15),
    pady=8,
    ipadx=24,
    ipady=6,
    row=7,
    column=2,
    columnspan=2,
    sticky=tk.W+tk.E+tk.N+tk.S
)
```

```
self.reset_btn = tk.Button(
    root,
    text="RESET",
    command=self.reset_callback,
    bg="#aaaaaa",
    fg="#ffffff",
    bd=2,
    relief=tk.FLAT
)
self.reset_btn.grid(
    padx=15,
    pady=(4, 12),
    ipadx=24,
    ipady=6,
    row=8,
    column=0,
    columnspan=4,
    sticky=tk.W+tk.E+tk.N+tk.S
)

self.status_label = tk.Label(
    root,
    textvariable=self._status,
    bg="#eeeeee",
    anchor=tk.W,
    justify=tk.LEFT,
    relief=tk.FLAT,
    wraplength=350
)
self.status_label.grid(
    padx=12,
    pady=(0, 12),
    ipadx=0,
    ipady=1,
    row=9,
    column=0,
    columnspan=4,
    sticky=tk.W+tk.E+tk.N+tk.S
)

tk.Grid.columnconfigure(root, 0, weight=1)
tk.Grid.columnconfigure(root, 1, weight=1)
tk.Grid.columnconfigure(root, 2, weight=1)
```

```
        tk.Grid.columnconfigure(root, 3, weight=1)

    def selectfile_callback(self):
        try:
            name = filedialog.askopenfile()
            self._file_url.set(name.name)
            # print(name.name)
        except Exception as e:
            self._status.set(e)
            self.status_label.update()

    def freeze_controls(self):
        self.file_entry.configure(state="disabled")
        self.key_entry.configure(state="disabled")
        self.select_btn.configure(state="disabled")
        self.encrypt_btn.configure(state="disabled")
        self.decrypt_btn.configure(state="disabled")
        self.reset_btn.configure(text="CANCEL", command=self.cancel_callback,
            fg="#ed3833", bg="#fafafa")
        self.status_label.update()

    def unfreeze_controls(self):
        self.file_entry.configure(state="normal")
        self.key_entry.configure(state="normal")
        self.select_btn.configure(state="normal")
        self.encrypt_btn.configure(state="normal")
        self.decrypt_btn.configure(state="normal")
        self.reset_btn.configure(text="RESET", command=self.reset_callback,
            fg="#ffffff", bg="#aaaaaa")
        self.status_label.update()

    def encrypt_callback(self):
        self.freeze_controls()

        try:
            self._cipher = EncryptionTool(
                self._file_url.get(),
                self._secret_key.get(),
                self._salt.get()
            )
            for percentage in self._cipher.encrypt():
                if self.should_cancel:
                    break
                percentage = "{0:.2f}%".format(percentage)
                self._status.set(percentage)
                self.status_label.update()
            self._status.set("File Encrypted!")
            if self.should_cancel:
                self._cipher.abort()
                self._status.set("Cancelled!")
            self._cipher = None
            self.should_cancel = False
        except Exception as e:
            # print(e)
            self._status.set(e)

        self.unfreeze_controls()

    def decrypt_callback(self):
        self.freeze_controls()

        try:
            self._cipher = EncryptionTool(
                self._file_url.get(),
                self._secret_key.get(),
                self._salt.get()
            )
            for percentage in self._cipher.decrypt():
                if self.should_cancel:
                    break
                percentage = "{0:.2f}%".format(percentage)
                self._status.set(percentage)
                self.status_label.update()
            self._status.set("File Decrypted!")
            if self.should_cancel:
                self._cipher.abort()
                self._status.set("Cancelled!")
            self._cipher = None
            self.should_cancel = False
        except Exception as e:
            # print(e)
            self._status.set(e)

        self.unfreeze_controls()

    def reset_callback(self):
        self._cipher = None
        self._file_url.set("")
        self._secret_key.set("")
        self._salt.set("")
        self._status.set("---")

    def cancel_callback(self):
        self.should_cancel = True

    def show_help_callback(self):
        messagebox.showinfo(
            "How To",
            """1. Open the App and Click SELECT FILE Button and select your file e.g. "abc.jpg".
2. Enter your Secret Key (This can be any alphanumeric letters). Remember this so you can Decrypt the file later.
3. Click ENCRYPT Button to encrypt. A new encrypted file with ".kryp" extention e.g. "abc.jpg.kryp" will be created in the same directory where the "abc.jpg" is.
4. When you want to Decrypt a file you, will select the file with the ".kryp" extention and Enter your Secret Key which you chose at the time of Encryption. Click DECRYPT Button to decrypt. The decrypted file will be of the same name as before with the suffix "__dekrypted__" e.g. "abc__dekrypted__.jpg".
5. Click RESET Button to reset the input fields and status bar.
6. You can also Click CANCEL Button during Encryption/Decryption to stop the process."""
        )


if __name__ == "__main__":
    ROOT = tk.Tk()
    MAIN_WINDOW = MainWindow(ROOT)
    ROOT.mainloop()
```