Nguyễn Văn Hưng – 23520569

import ctypes

from ctypes import c\_char\_p, c\_ubyte, POINTER, byref, create\_string\_buffer

import os

import argparse

os.add\_dll\_directory("C:\\msys64\\mingw64\\bin")

AES\_KEY\_LEN = 16

AES\_BLOCK\_SIZE = 16

# Load DLL

dll = ctypes.CDLL("./AES.so")

# Define functions

dll.GenerateAESKeyIV.argtypes = [ctypes.POINTER(ctypes.c\_ubyte), ctypes.POINTER(ctypes.c\_ubyte)]

dll.GenerateAESKeyIV.restype = None

dll.SaveKeyToFile.argtypes = [ctypes.c\_char\_p,

                              ctypes.POINTER(ctypes.c\_ubyte),

                              ctypes.POINTER(ctypes.c\_ubyte)]

dll.SaveKeyToFile.restype = None

dll.LoadKeyFromFile.argtypes = [ctypes.c\_char\_p,

                                ctypes.POINTER(ctypes.c\_ubyte),

                                ctypes.POINTER(ctypes.c\_ubyte)]

dll.LoadKeyFromFile.restype = None

dll.AESEncryptFile.argtypes = [ctypes.POINTER(ctypes.c\_ubyte),

                            ctypes.POINTER(ctypes.c\_ubyte),

                            ctypes.c\_char\_p,

                            ctypes.c\_char\_p]

dll.AESEncryptFile.restype = None

dll.AESDecryptFile.argtypes = [ctypes.POINTER(ctypes.c\_ubyte),

                            ctypes.POINTER(ctypes.c\_ubyte),

                            ctypes.c\_char\_p,

                            ctypes.c\_char\_p]

dll.AESDecryptFile.restype = None

# --- Helper functions ---

def allocate\_key\_iv():

    return (c\_ubyte \* AES\_KEY\_LEN)(), (c\_ubyte \* AES\_BLOCK\_SIZE)()

def generate\_and\_save(keyfile: str):

    key, iv = allocate\_key\_iv()

    dll.GenerateAESKeyIV(key, iv)

    dll.SaveKeyToFile(keyfile.encode(), key, iv)

    print(f"Key and IV saved to '{keyfile}'")

def load\_key\_iv(keyfile: str):

    key, iv = allocate\_key\_iv()

    dll.LoadKeyFromFile(keyfile.encode(), key, iv)

    return key, iv

def show\_key(keyfile: str):

    key, iv = load\_key\_iv(keyfile)

    print("Key:", bytes(key).hex().upper())

    print("IV :", bytes(iv).hex().upper())

def encrypt\_file(keyfile: str, infile: str, outfile: str):

    key, iv = load\_key\_iv(keyfile)

    dll.AESEncryptFile(key, iv, infile.encode(), outfile.encode())

    print(f"Encrypted '{infile}' → '{outfile}'")

def decrypt\_file(keyfile: str, infile: str, outfile: str):

    key, iv = load\_key\_iv(keyfile)

    dll.AESDecryptFile(key, iv, infile.encode(), outfile.encode())

    print(f"Decrypted '{infile}' → '{outfile}'")

def main():

    parser = argparse.ArgumentParser(description="AES Encrypt/Decrypt with C DLL backend")

    subparsers = parser.add\_subparsers(dest="command")

    # generate

    p\_gen = subparsers.add\_parser("generate", help="Generate AES key and IV")

    p\_gen.add\_argument("keyfile")

    # show

    p\_show = subparsers.add\_parser("show", help="Display AES key and IV")

    p\_show.add\_argument("keyfile")

    # encrypt

    p\_enc = subparsers.add\_parser("encrypt", help="Encrypt file")

    p\_enc.add\_argument("keyfile")

    p\_enc.add\_argument("infile")

    p\_enc.add\_argument("outfile")

    # decrypt

    p\_dec = subparsers.add\_parser("decrypt", help="Decrypt file")

    p\_dec.add\_argument("keyfile")

    p\_dec.add\_argument("infile")

    p\_dec.add\_argument("outfile")

    args = parser.parse\_args()

    if args.command == "generate":

        generate\_and\_save(args.keyfile)

    elif args.command == "show":

        show\_key(args.keyfile)

    elif args.command == "encrypt":

        encrypt\_file(args.keyfile, args.infile, args.outfile)

    elif args.command == "decrypt":

        decrypt\_file(args.keyfile, args.infile, args.outfile)

    else:

        parser.print\_help()

if \_\_name\_\_ == "\_\_main\_\_":

    main()

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.