Python version: 3.6.5

Objective: The aim of this program is to allow users to choose whether they wish to encrypt a message or decrypt an existing encrypted text file.

The encryption section of the program uses AES-256-CBC encryption algorithm to encrypt a message from the user’s input, using either the user’s input as a key, or generating a random key. The encrypted message is then saved to a text file with a filename as specified by the user.

The decryption section of the program uses the same AES-256-CBC algorithm to decrypt a chosen text file specified by the user. The algorithm uses a key determined from the user’s input of the original key used during the encryption process, and then presents the resulting unencrypted message for the user.

Import the required modules

Print a welcome message for the user

Set the rest of the program within a loop so the user can perform an action multiple times

Print the available options (encrypt/decrypt/quit), indicating the letter in brackets that initiates each option

Assign the user input to a variable *“choice”*

If *“choice”* is the letter ‘e’

Set this section of the encryption program within a loop so the user can perform an action multiple times

Print the available options (manual key/random key/quit), indicating the letter in brackets that initiates each option

Assign the user input to a variable *“choice”*

If *“choice”* is the letter ‘k’

Implement inputKey function to prompt user for a key, which must be 32 bytes in length, then assign the input to a variable *“passphrase”*

Print the available option (view key = yes/no), indicating the letter in brackets that initiates each option

Assign the user input to a variable *“choice”*

If *“choice”* is the letter ‘y’

Print message and display user input *“passphrase”*

Else

Print warning message that saving the key is recommended and provide option (view key = yes/no), indicating the letter in brackets that initiates each option

Assign the user input to a variable *“choice”*

If not *“choice”* is the letter ‘y’

Print message and display user input *“passphrase”*

If *“choice”* is the letter ‘r’

Implement randomKey function to generate a random key, which is 32 bytes in length, then assign the input to a variable *“passphrase”*

Print the available option (view key = yes/no), indicating the letter in brackets that initiates each option

Assign the user input to a variable *“choice”*

If *“choice”* is the letter ‘y’

Print message and display randomly generated *“passphrase”*

Else

Print warning message that saving the key is recommended and provide option (view key = yes/no), indicating the letter in brackets that initiates each option

Assign the user input to a variable *“choice”*

If not *“choice”* is the letter ‘y’

Print message and display randomly generated *“passphrase”*

Else if *“choice”* is the letter ‘q’

Break the loop to return to menu

Else

Print a warning message stating that the letter choice was invalid

Continue back to the beginning of the loop to prompt for a valid input

Set this next section of the encryption program within a loop so the user can perform an action multiple times

Implement userInput function to prompt user for a message, with no length requirements, then assign to a variable *“message”*

Print the available option (view message/new message/continue), indicating the letter in brackets that initiates each option

Assign the user input to a variable *“choice”*

If *“choice”* is the letter ‘e’

Continue back to the beginning of the loop to prompt for a different message

Else if *“choice”* is the letter ‘v’

Print message and display *“message”*

Print the available option (new message/continue), indicating the letter in brackets that initiates each option

Assign the user input to a variable *“choice”*

If *“choice”* is the letter ‘e’

Continue back to the beginning of the loop to prompt for a different message

Else

Break the loop to continue to the next section of the program

Else

Break the loop to continue to the next section of the program

Print a message to inform user to message is being encrypted

Implement encrypt function to encrypt the message from the user’s input, then assign to a variable *“hexcipher”*

Set this next section of the encryption program within a loop so the user can perform an action multiple times

Implement userInput function to prompt user for a filename, with no length requirements, then assign to a variable *“filename”*

Print the available option (new filename/continue), indicating the letter in brackets that initiates each option

Assign the user input to a variable *“choice”*

If *“choice”* is the letter ‘e’

Continue back to the beginning of the loop to prompt for a different filename

Else

Implement saveFile function to write the encrypted output from the encrypt function to a text file, with the filename from the user’s input, then assign to a variable *“encrypted\_file”*

Print message to inform user the file has been saved

Else if *“choice”* is the letter ‘d’

Print the available option (return to menu/continue), indicating the letter in brackets that initiates each option

Assign the user input to a variable *“choice”*

If *“choice”* is the letter ‘q’

Break the loop to return to the main menu of the program

Else

Set this section of the decryption program within a loop so the user can perform an action multiple times

Implement userInput function to prompt user for a filename, with no length requirements, then assign to a variable *“en\_filename”*

If not an existing filename found

Print warning message that the file does not exist and prompt for a different filename

Continue back to the beginning of the loop to prompt for a different filename

Else

Print a message to inform the user to file has been located

Open the specified file in read mode

Assign the data from the specified file to a variable *“encrypted\_file”*

Implement userInput function to prompt user for the original key used during encryption, then assign to a variable *“keyphrase”*

Implement decrypt function to decrypt the message from the specified file, then assign to a variable *“plaintext”*

Print message and display *“plaintext”*

Else if *“choice”* is the letter ‘q’

Print farewell message

Break the loop to end the program

Else

Print a warning message stating that the letter choice was invalid

Continue back to the beginning of the loop to prompt for a valid input

encrypt(passphrase, message) function:

Assign imported default\_backend() to variable *“backend”*

Generate a random 16-byte initialisation vector, assigned to the variable *“iv”*

Encode the user’s input or randomly generated passphrase with utf-8 encoding, assigned to the variable *“key”*

Encode the message from the user’s input with utf-8 encoding, assigned to the variable *“en\_plaintext”*

Generate a random 1-byte padding character, assigned to the variable *“pad\_char”*

Calculate from the length of *“en\_plaintext”* how many characters remain from a multiple of 16, as well as how much padding will be needed to reach a multiple of 16, then assign to a variable *“padsize”*

If from the length of *“en\_plaintext”,* the amount of remaining characters from a multiple of 16 is greater than 0

Concatenate *“pad\_char”* \* *“padsize”* to *“en\_plaintext”*, then assign to a variable *“pad\_text”*

Else assign the variable *“en\_plaintext”* to a variable *“pad\_text”*

Create cipher object using imported Cipher module and variables *“key”*, *“iv”*, and *“backend”*, then assign to a variable *“cipher”*

Create encryptor with variable *“cipher”*, assigned to the variable *“encryptor”*

Pass the *“pad\_text”* through the encryptor and assign to the variable *“ciphertext”*

Concatenate *“iv”* and *“pad\_char”* to end of *“ciphertext”*, then assign to a variable *“cipher\_iv\_pad”*

Change *“cipher\_iv\_pad”* from bytes to hex, assigned to the variable *“hexcipher”*

Return the variable *“hexcipher”*

decrypt(keyphrase, encrypted\_file) function:

Assign imported default\_backend() to variable *“backend”*

Encode the user’s input of a passphrase with utf-8 encoding, assigned to the variable *“key”*

Change *“encrypted\_file”* from hex to bytes, assigned to the variable *“ciphertext”*

Retrieve the iv from the end of the *“ciphertext”* and assign to the variable *“iv”*

Retrieve the padding character from the end of the *“ciphertext”* and assign to the variable *“pad\_char”*

Assign the remaining encrypted data, without the concatenated *“iv”* and *“pad\_char”*, to the variable *“ciphertext”*

Set this section of the function within a loop so the user can perform an action multiple times

Try

Create cipher object using imported Cipher module and variables *“key”*, *“iv”*, and *“backend”*, then assign to a variable *“cipher”*

Except ValueError

Implement userInput function to print a warning message that the input was not the correct key, and prompt the user for a different passphrase, assigning the input to the variable *“key”*

Encode the user’s input of a passphrase with utf-8 encoding, assigned to the variable *“key”*

Continue back to the beginning of the loop to attempt the cipher creation with the new key

Create decryptor with variable *“cipher”*, assigned to the variable *“decryptor”*

Pass the *“ciphertext”* through the decryptor and assign to the variable *“plaintext”*

Calculate the amount of padding that was used at the end of the original message

If the amount of padding is 0, assign *“plaintext”* to the variable *“unpad\_text”*

Else remove the padding from the end of *“plaintext”*, then assign to the variable *“unpad\_text”*

Decode *“unpad\_text”* with utf-8 decoding, assigned to the variable *“plaintext”*

Return the variable *“plaintext”*

saveFile(filename) function:

If the filename exists in the current folder

Print a warning message that the file exists and will be overwritten, then give the option (continue = yes/no), indicating the letter in brackets that initiates each option

Assign the user input to a variable *“choice”*

Set this section of the function within a loop so the user can perform an action multiple times

If *“choice”* is the letter ‘y’

Break the loop to continue saving the file

Else

Implement userInput function to prompt user for a different input for a filename, then assign to a variable *“filename”*

Open a text document with that *“filename”*, write the data to the file, and then close it

Open a text document with that *“filename”*, write the data to the file, and then close it

userInput(prompt) function:

Set the whole function within a loop so it can be restarted until the appropriate input is given

Prompt the user to input a character which is assigned to the variable “*value*” and strip it of its whitespace

If the length of *“value”* is one character or more

Return the value

Break the loop of the function

Else

Print an error message stating that at least one character is required as input

Allow the function to continue its loop until the user inputs at least one character of text

inputKey(prompt) function:

Set the whole function within a loop so the user can perform an action multiple times

While True

Prompt the user to input a character which is assigned to the variable “*value*” and strip it of its whitespace

Encode the *“value”* with utf-8 encoding and assign to the variable *“encoded\_key”*

Calculate the byte length of *“encoded\_key”* and assign to the variable *“key\_length”*

If the key length is 32 bytes

Return *“value”*

Else

Print a message that shows the length of the current input and prompt the user for an input that is 32-bytes

Allow the function to continue its loop until the user inputs a 32-byte passphrase

randomKey(size, characters) function:

Define the function with a size of 32-bytes, and including string characters of ascii\_uppercase, ascii\_lowercase and digits

Use the imported random module to select a random choice of any of the specified characters until the size of 32-bytes has been reached, then assign the data to the variable *“random\_key”*