# Author: Trevor Perrin

# See the LICENSE file for legal information regarding use of this file.

"""Pure-Python AES implementation."""

from .cryptomath import \*

from .aes import \*

from .rijndael import rijndael

def new(key, mode, IV):

return Python\_AES(key, mode, IV)

class Python\_AES(AES):

def \_\_init\_\_(self, key, mode, IV):

AES.\_\_init\_\_(self, key, mode, IV, "python")

self.rijndael = rijndael(key, 16)

self.IV = IV

def encrypt(self, plaintext):

AES.encrypt(self, plaintext)

plaintextBytes = plaintext[:]

chainBytes = self.IV[:]

#CBC Mode: For each block...

for x in range(len(plaintextBytes)//16):

#XOR with the chaining block

blockBytes = plaintextBytes[x\*16 : (x\*16)+16]

for y in range(16):

blockBytes[y] ^= chainBytes[y]

#Encrypt it

encryptedBytes = self.rijndael.encrypt(blockBytes)

#Overwrite the input with the output

for y in range(16):

plaintextBytes[(x\*16)+y] = encryptedBytes[y]

#Set the next chaining block

chainBytes = encryptedBytes

self.IV = chainBytes[:]

return plaintextBytes

def decrypt(self, ciphertext):

AES.decrypt(self, ciphertext)

ciphertextBytes = ciphertext[:]

chainBytes = self.IV[:]

#CBC Mode: For each block...

for x in range(len(ciphertextBytes)//16):

#Decrypt it

blockBytes = ciphertextBytes[x\*16 : (x\*16)+16]

decryptedBytes = self.rijndael.decrypt(blockBytes)

#XOR with the chaining block and overwrite the input with output

for y in range(16):

decryptedBytes[y] ^= chainBytes[y]

ciphertextBytes[(x\*16)+y] = decryptedBytes[y]

#Set the next chaining block

chainBytes = blockBytes

self.IV = chainBytes[:]

return ciphertextBytes