

# **Encryption & Decryption Using AES Algorithm**

## **Encryption & Decryption Using AES Algorithm**

### **What Is AES**

- There are dozens of encryption algorithms out there, but the most commonly used is AES—short for Advanced Encryption Standard, also known as Rijndael.
- AES is an encoding algorithm that transforms plain text data into a version known as ciphertext that's not possible for humans or machines to understand without an encryption key—a password.
- For example, you use AES in software development to securely store passwords in a database.
- Storing a password as plain text would allow anyone with access to the database to log in to user accounts, so encrypting them is the first step to adding a layer of security to your authentication system.

### **Symmetric Encryptions**

- Symmetric encryption uses a single key to encrypt and decrypt. If you encrypt a zip file, then decrypt with the same key, you are using symmetric encryption.
- Symmetric encryption is also called “secret key” encryption because the key must be kept secret from third parties.
- Symmetric-key algorithms are algorithms for cryptography that use the same cryptographic keys for both the encryption of plaintext and the decryption of ciphertext.

## **Asymmetric Encryptions**

- Asymmetric encryption (also known as asymmetric cryptography) allows users to encrypt information using shared keys.
- You need to send a message across the internet, but you don't want anyone but the intended recipient to see what you've written.
- Public-key cryptography, or asymmetric cryptography, is the field of cryptographic systems that use pairs of related keys.
- Key pairs are generated with cryptographic algorithms based on mathematical problems termed one-way functions.

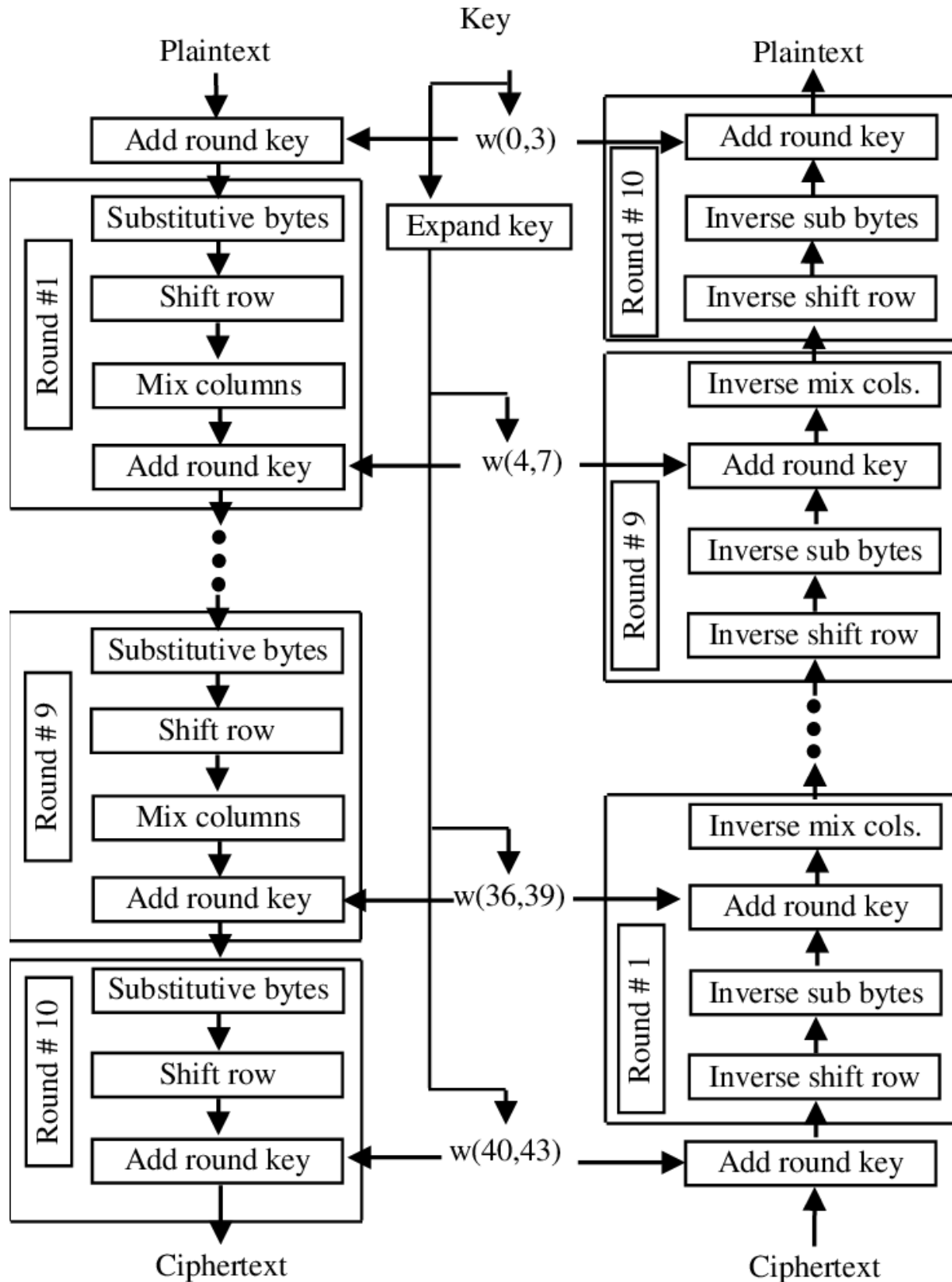
## **Symmetric vs Asymmetric Encryption**

- AES is a symmetric-key algorithm, meaning the same key (aka passphrase or password) is used to encrypt and decrypt the data.
- This characteristic presents pros and cons detailed in the following sections.
- Asymmetric methods use a public key for encryption and a secret key for decryption.
- Anyone can send encrypted messages but only the receiver knows how to decrypt them.
- TLS certificates used for secure HTTP communication (HTTPS) leverage asymmetric encryption, for example.

The AES algorithm can be divided into 3 main parts:

- Generate a key
- Generate a cipher
- Encrypt/decrypt the data with the cipher

## Block Diagram



The AES algorithm consist of sequences of 128 bits for each input and output. Sometimes these sequences will be referred to as blocks where the number of bits they include, will be referred to as their length. The AES algorithm cipher key is a sequence of 128, 192 or 256 bits. Ten rounds are on AES algorithm encryption, as can be shown in Fig. (1). Initially, the 128-bit key is expanded into eleven so-called round keys, each of them 128 bits in size. To ensure high security of the encryption each round contain transformation by utilize the corresponding key of cipher.

### Generating the AES Key

- AES requires a secret passphrase known as a “*key*” to encrypt/decrypt data.
- Anybody with the key can decrypt your data, so you need it to be strong and hidden from everyone—only the software program should be able to access it.
- The key can be either 128, 192, 256, or 512 bit long. An AES cipher using a 512-bit key is abbreviated as AES 512, for example.
- The longer the key, the more secure it is, but the slower the encryption/decryption will be as well. 128 bits is equivalent to 24 characters in base64 encoding, 44 characters for 256 bits.
- Since storage space isn’t usually a problem and the difference in speed between the versions is negligible, a good rule of thumb is to use 256-bit keys.

### Generating the Cipher

- The cipher is **the algorithm used to perform encryption/decryption.**
- To handle data of arbitrary length and nature, several modes of operation for AES are possible.

Each mode changes the general algorithm while offering pros and cons.

## Encrypting/Decrypting data

- AES operates on data blocks of 16 bytes represented as 4x4 two-dimensional arrays.
- The byte content of these matrices is transformed using a mathematical function defined by the cipher (also called *block cipher*), which gives us a ciphertext—the encrypted result.

Without going into the technical details, the mathematical function guarantees the strength of the encryption, which is why **AES is considered unbreakable as long as the properties of the function are respected**—strong key, correctly implemented cipher, unique nonce, unique initialization vector, etc

- The AES key is the weakest link in the chain: you need to protect it at all costs!
- While having your AES key in a .env file is a simple way to go about it, with a secret manager like Onboardbase you can inject AES passphrases in software programs as environment variables in your CI/CD process in just 3 commands.

## Implementing AES in Python

Fortunately, we don't have to implement AES from scratch, but you can give it a try if you're feeling spicy. In order to avoid doing so, we first need to install the **pycrypto** library, which can be done via pip with the following command:

```
pip install pycrypto
```

which should run without errors. Once **pycrypto** is installed, create a python file and write the following to import everything we need:

```
1 import hashlib
2 from Crypto import Random
3 from Crypto.Cipher import AES
4 from base64 import b64encode, b64decode
```

Now we are going to create a class for our AES cipher with the following constructor:

```
1 class AESCipher(object):
2     def __init__(self, key):
3         self.block_size = AES.block_size
4         self.key = hashlib.sha256(key.encode()).digest()
```

Lets take a quick walk through the constructor, it receives a key which can be of any length. Then we proceed to generate a 256 bit hash from that key. A hash is basically a unique identifier of a given length, 32 characters in this case, of any input no matter its length. This enables that you can pass the constructor your name or a phrase, and it will generate an unique 256 bit key for your cipher. We also set `block_size` to 128, which is the block size of AES.

Before we proceed to define the `encrypt` and `decrypt` methods for our `AESCipher` class, lets first create the `__pad` and `__unpad` methods. Also, beginners in Python can learn from the [best Python tutorials](#) to enhance their learning.

## Pad

The `__pad` method receives the `plain_text` to be encrypted and adds a number bytes for the text to be a multiple of 128 bits. This number is stored in `number_of_bytes_to_pad`. Then in `ascii_string` we generate our padding character, and `padding_str` will contain that character times `number_of_bytes_to_pad`. So we only have to add `padding_str` at the end of our *plain\_text* so that it is now a multiple of 128 bits.

```
1     def __pad(self, plain_text):
2         number_of_bytes_to_pad = self.block_size - len(plain_text) % self.block_size
3         ascii_string = chr(number_of_bytes_to_pad)
4         padding_str = number_of_bytes_to_pad * ascii_string
5         padded_plain_text = plain_text + padding_str
6         return padded_plain_text
7
```

## Unpad

In an opposite manner, `__unpad` method will receive the decrypted text, also known as `plain_text` and will remove all the extra added characters in the `__pad` method. For that we first must identify the last character and store in `bytes_to_remove` how many bytes we need to trim of the end of `plain_text` in order to unpad it.



```
1     @staticmethod
2     def __unpad(plain_text):
3         last_character = plain_text[len(plain_text) - 1:]
4         bytes_to_remove = ord(last_character)
5         return plain_text[:-bytes_to_remove]
6
```

## Encrypt

With these two methods out of the way, lets implement the `encrypt` method

The `encrypt` method receives the `plain_text` to be encrypted. First we pad that `plain_text` in order to be able to encrypt it. After we generate a new random `iv` with the size of an AES block, 128bits. We now create our AES cipher with `AES.new` with our `key`, in mode `CBC` and with our just generated `iv`. We now invoke the `encrypt` function of our cipher, passing it our `plain_text` converted to bits. The encrypted output is then placed after our `iv` and converted back from bits to readable characters.

```
1     def encrypt(self, plain_text):
2         plain_text = self.__pad(plain_text)
3         iv = Random.new().read(self.block_size)
4         cipher = AES.new(self.key, AES.MODE_CBC, iv)
5         encrypted_text = cipher.encrypt(plain_text.encode())
6         return b64encode(iv + encrypted_text).decode("utf-8")
```

## Decrypt

Now that we can encrypt text, but I'm sure we would like to be able to decrypt it:

In order to decrypt, we must backtrack all the steps done in the `encrypt` method.

First we convert our `encrypted_text` to bits and extract the `iv`, which will be the first 128 bits of our `encrypted_text`. Much like before, we now create a new AES cipher with our key, in mode CBC and with the extracted `iv`. We now invoke the `decrypt` method of our cipher and convert it to text from bits. We remove the padding with `__unpad` and that's it!

```
1     def decrypt(self, encrypted_text):
2         encrypted_text = b64decode(encrypted_text)
3         iv = encrypted_text[:self.block_size]
4         cipher = AES.new(self.key, AES.MODE_CBC, iv)
5         plain_text = cipher.decrypt(encrypted_text[self.block_size:]).decode("utf-8")
6         return self.__unpad(plain_text)
```

**Code:**

```
from Crypto import Random
from Crypto.Cipher import AES
import os
import os.path
from os import listdir
from os.path import isfile, join
import time

class Encryptor:
    def __init__(self, key):
        self.key = key

    def pad(self, s):
        return s + b"\0" * (AES.block_size - len(s) % AES.block_size)

    def encrypt(self, message, key, key_size=256):
        message = self.pad(message)
        iv = Random.new().read(AES.block_size)
        cipher = AES.new(key, AES.MODE_CBC, iv)
        return iv + cipher.encrypt(message)

    def encrypt_file(self, file_name):
        with open(file_name, 'rb') as fo:
            plaintext = fo.read()
```

```
enc = self.encrypt(plaintext, self.key)
with open(file_name + ".enc", 'wb') as fo:
    fo.write(enc)
os.remove(file_name)
```

```
def decrypt(self, ciphertext, key):
    iv = ciphertext[:AES.block_size]
    cipher = AES.new(key, AES.MODE_CBC, iv)
    plaintext = cipher.decrypt(ciphertext[AES.block_size:])
    return plaintext.rstrip(b"\0")
```

```
def decrypt_file(self, file_name):
    with open(file_name, 'rb') as fo:
        ciphertext = fo.read()
    dec = self.decrypt(ciphertext, self.key)
    with open(file_name[:-4], 'wb') as fo:
        fo.write(dec)
    os.remove(file_name)
```

```
def getAllFiles(self):
    dir_path = os.path.dirname(os.path.realpath(__file__))
    dirs = []
    for dirName, subdirList, fileList in os.walk(dir_path):
        for fname in fileList:
            if (fname != 'script.py' and fname != 'data.txt.enc'):
```

```

        dirs.append(dirName + "\\\" + fname)
    return dirs

def encrypt_all_files(self):
    dirs = self.getAllFiles()
    for file_name in dirs:
        self.encrypt_file(file_name)

def decrypt_all_files(self):
    dirs = self.getAllFiles()
    for file_name in dirs:
        self.decrypt_file(file_name)

key =
b'[EX\xc8\xd5\xbf{\xa2$\x05(\xd5\x18\xbf\xc0\x85)\x10nc\x94\x02)j\xdf\xcb\x
4\x94\x9d(\x9e'
enc = Encryptor(key)
clear = lambda: os.system('cls')

if os.path.isfile('data.txt.enc'):
    while True:
        password = str(input("Enter password: "))
        enc.decrypt_file("data.txt.enc")
        p = "
        with open("data.txt", "r") as f:
            p = f.readlines()

```

```
if p[0] == password:  
    enc.encrypt_file("data.txt")  
    break
```

```
while True:
```

```
    clear()
```

```
    choice = int(input(
```

```
        "1. Press '1' to encrypt file.\n2. Press '2' to decrypt file.\n3. Press '3' to  
Encrypt all files in the directory.\n4. Press '4' to decrypt all files in the  
directory.\n5. Press '5' to exit.\n"))
```

```
    clear()
```

```
    if choice == 1:
```

```
        enc.encrypt_file(str(input("Enter name of file to encrypt: ")))
```

```
    elif choice == 2:
```

```
        enc.decrypt_file(str(input("Enter name of file to decrypt: ")))
```

```
    elif choice == 3:
```

```
        enc.encrypt_all_files()
```

```
    elif choice == 4:
```

```
        enc.decrypt_all_files()
```

```
    elif choice == 5:
```

```
        exit()
```

```
    else:
```

```
        print("Please select a valid option!")
```

```
else:
```

```
    while True:
```

```
clear()

password = str(input("Setting up stuff. Enter a password that will be used for
decryption: "))

repassword = str(input("Confirm password: "))

if password == repassword:
    break
else:
    print("Passwords Mismatched!")

f = open("data.txt", "w+")
f.write(password)
f.close()

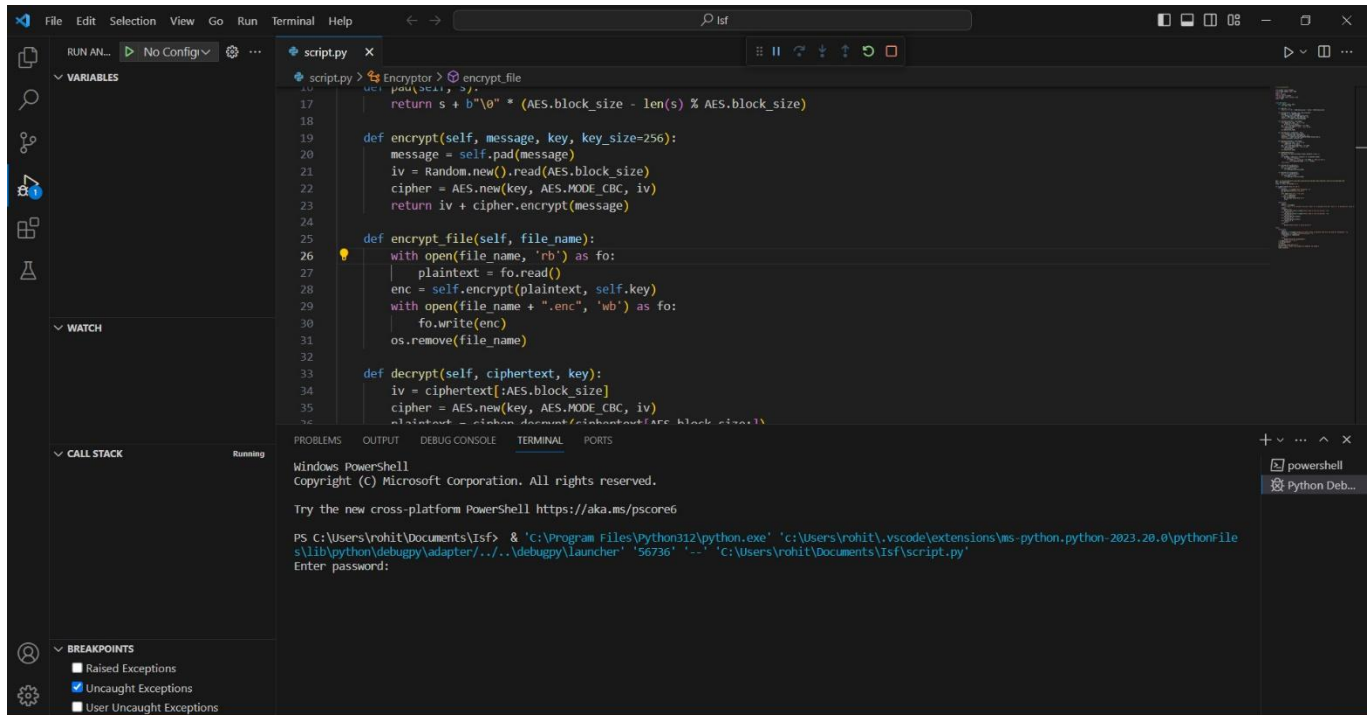
enc.encrypt_file("data.txt")

print("Please restart the program to complete the setup")

time.sleep(15)
```

## Output:

- Implementation of code

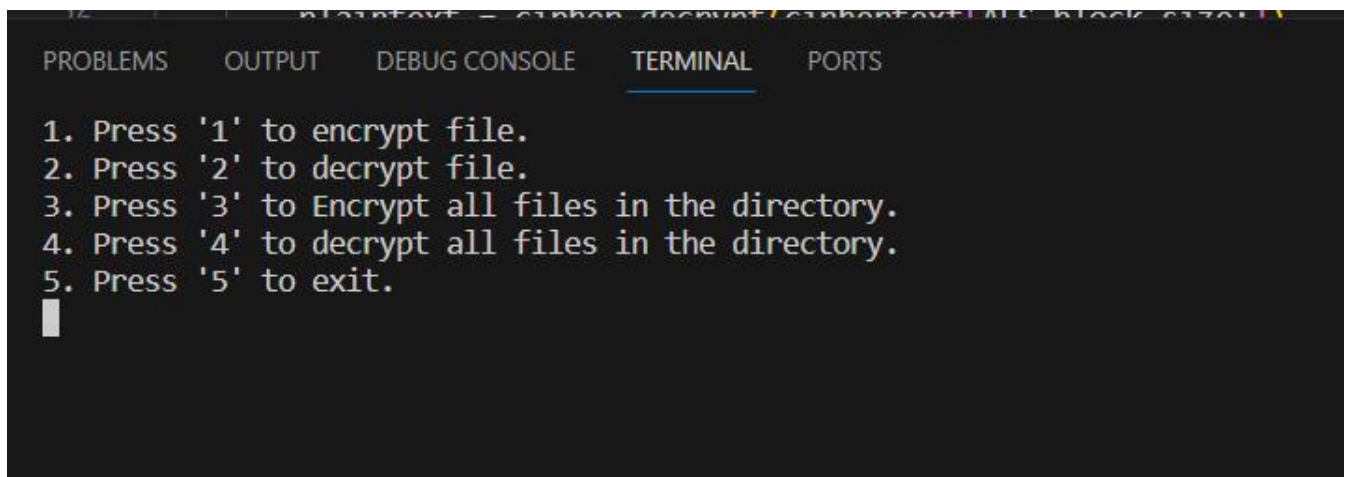


```
File Edit Selection View Go Run Terminal Help
script.py x
script.py > Encryptor > encrypt_file
17     return s + b"\0" * (AES.block_size - len(s) % AES.block_size)
18
19 def encrypt(self, message, key, key_size=256):
20     message = self.pad(message)
21     iv = Random.new().read(AES.block_size)
22     cipher = AES.new(key, AES.MODE_CBC, iv)
23     return iv + cipher.encrypt(message)
24
25 def encrypt_file(self, file_name):
26     with open(file_name, 'rb') as fo:
27         plaintext = fo.read()
28         enc = self.encrypt(plaintext, self.key)
29         with open(file_name + ".enc", 'wb') as fo:
30             fo.write(enc)
31         os.remove(file_name)
32
33 def decrypt(self, ciphertext, key):
34     iv = ciphertext[:AES.block_size]
35     cipher = AES.new(key, AES.MODE_CBC, iv)
36     plaintext = cipher.decrypt(ciphertext[AES.block_size:])
37
38 def decrypt_file(self, file_name):
39     with open(file_name, 'rb') as fo:
40         ciphertext = fo.read()
41         dec = self.decrypt(ciphertext, self.key)
42         with open(file_name[:-4], 'wb') as fo:
43             fo.write(dec)
44         os.remove(file_name)
45
46 if __name__ == '__main__':
47     obj = Encryptor()
48     while True:
49         print("\n1. Press '1' to encrypt file.")
50         print("2. Press '2' to decrypt file.")
51         print("3. Press '3' to Encrypt all files in the directory.")
52         print("4. Press '4' to decrypt all files in the directory.")
53         print("5. Press '5' to exit.")
54         choice = input("Enter your choice: ")
55         if choice == '1':
56             file_name = input("Enter file name: ")
57             obj.encrypt_file(file_name)
58             print("File encrypted successfully.")
59         elif choice == '2':
60             file_name = input("Enter file name: ")
61             obj.decrypt_file(file_name)
62             print("File decrypted successfully.")
63         elif choice == '3':
64             directory = input("Enter directory path: ")
65             obj.encrypt_directory(directory)
66             print("All files in the directory encrypted successfully.")
67         elif choice == '4':
68             directory = input("Enter directory path: ")
69             obj.decrypt_directory(directory)
70             print("All files in the directory decrypted successfully.")
71         elif choice == '5':
72             break
73         else:
74             print("Invalid choice. Please enter a valid option.")
```

Windows PowerShell  
Copyright (C) Microsoft Corporation. All rights reserved.  
Try the new cross-platform PowerShell <https://aka.ms/powershell>

PS C:\Users\rohit\Documents\Isf> & 'C:\Program Files\Python312\python.exe' 'C:\Users\rohit\.vscode\extensions\ms-python.python-2023.20.0\pythonFile  
s\lib\python\debugpy\adapter\..\..\debugpy\launcher' '56736' '-.' 'C:\Users\rohit\Documents\Isf\script.py'  
Enter password:

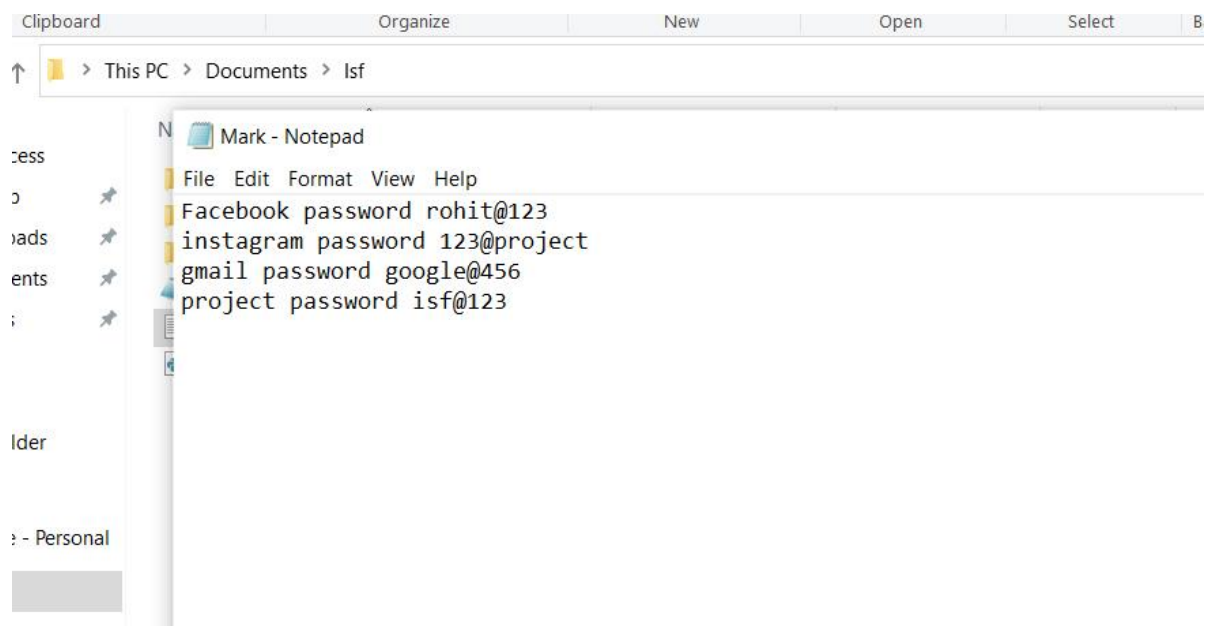
- Options



```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
1. Press '1' to encrypt file.
2. Press '2' to decrypt file.
3. Press '3' to Encrypt all files in the directory.
4. Press '4' to decrypt all files in the directory.
5. Press '5' to exit.
█
```



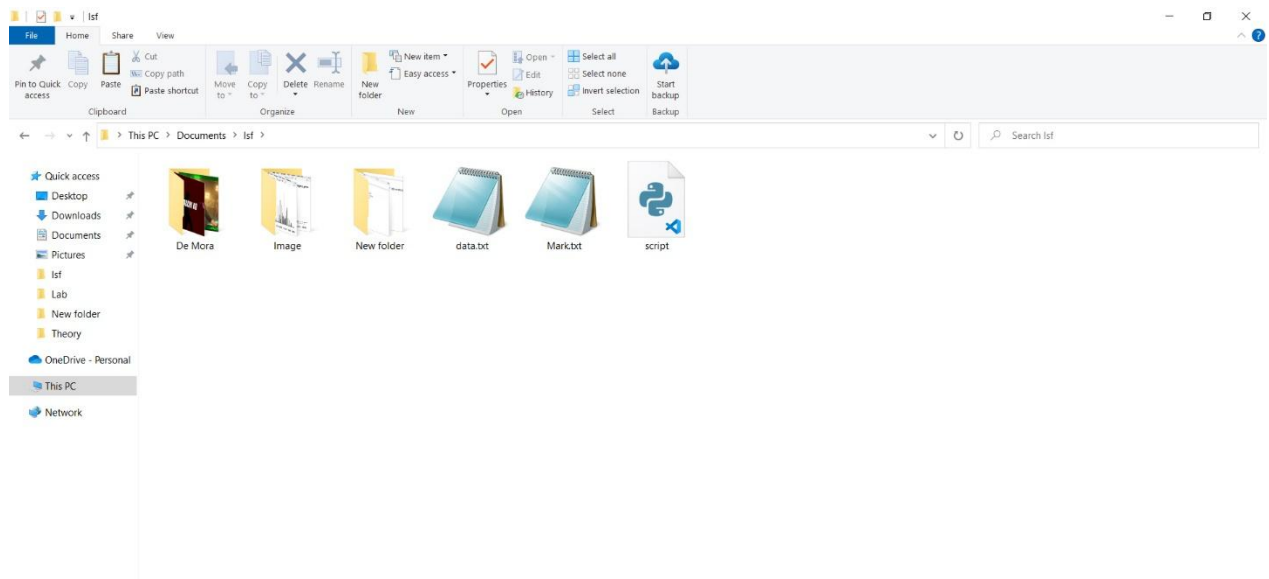
- File before Encryption



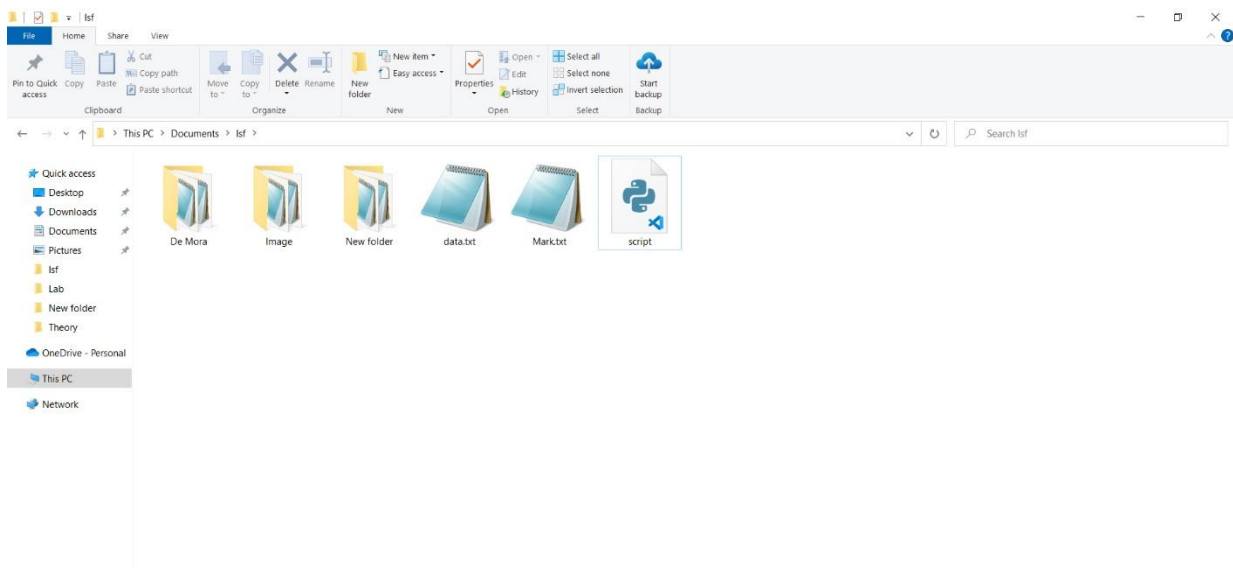
- File after Encryption



- Directory before Encryption

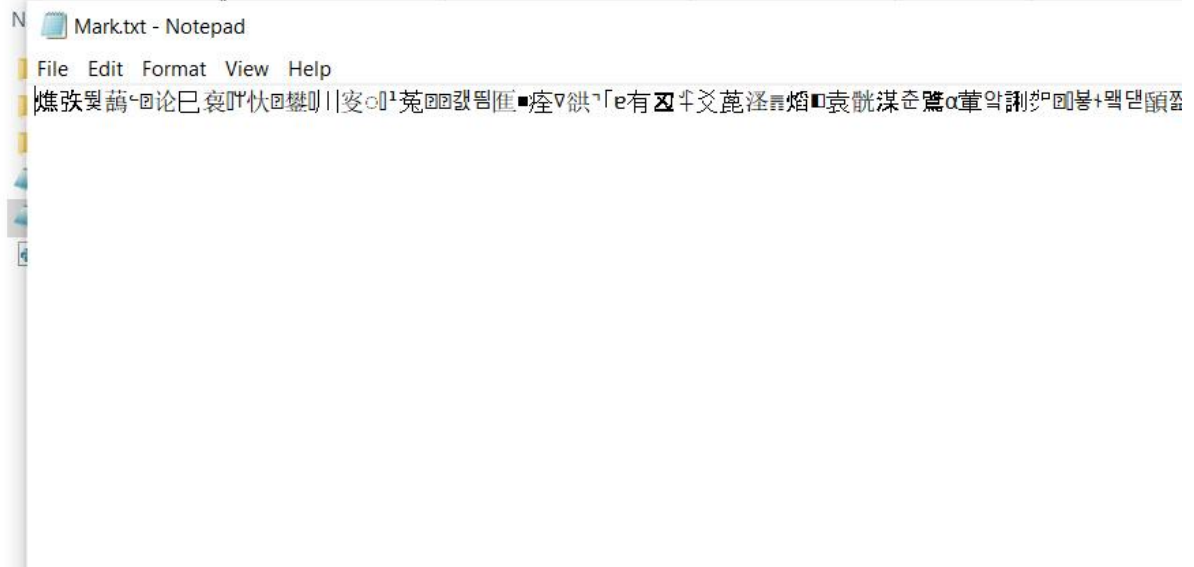


- Directory after Encryption

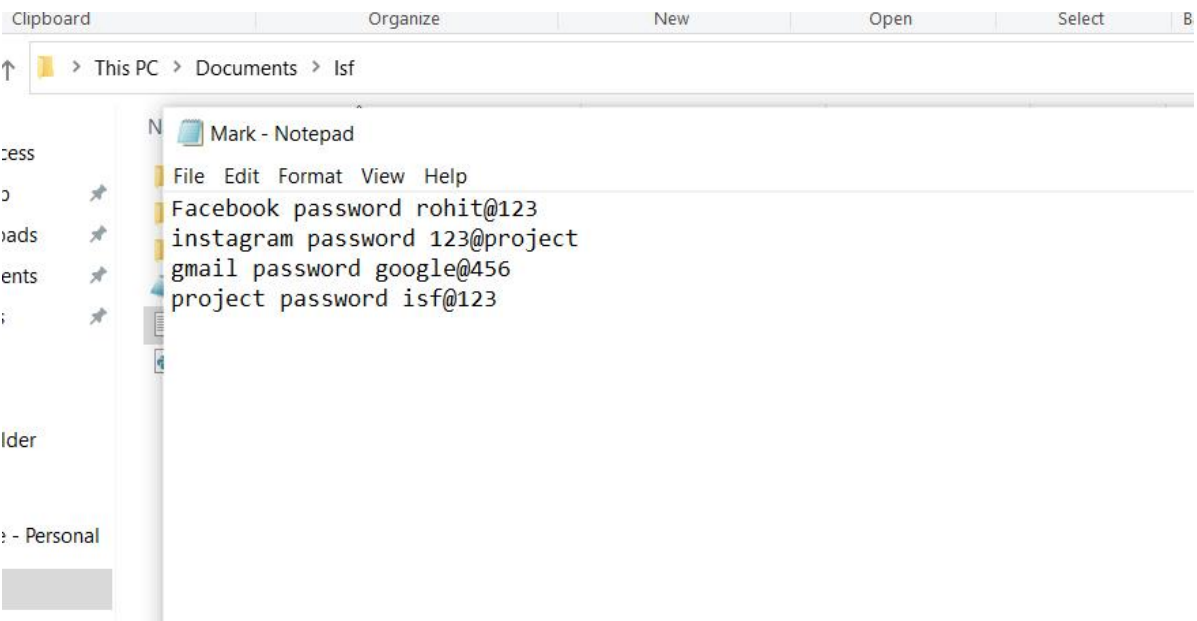


- File before Decryption

PC > Documents > Isf



- File after Decryption



**Conclusion:**

Advanced Encryption Standard (AES) algorithm is one of the most common and widely symmetric block cipher algorithm used worldwide. This algorithm has an own particular structure to encrypt and decrypt sensitive data and is applied in hardware and software all over the world. It is extremely difficult for hackers to get the real data when encrypting by AES algorithm. Till date is not any evidence to crack this algorithm. AES has the ability to deal with three different key sizes such as AES 128, 192 and 256 bit and each of these ciphers has 128 bit block size. This paper will provide an overview of AES algorithm and explain several crucial features of this algorithm in details and demonstrate some previous researches that have done on it with comparing to other algorithms such as DES, 3DES, Blowfish etc.