**Cryptography And Network Security Lab**

**Assignment submission**

**PRN No: 2019BTECS00017**

**Full name: Muskan Raju Attar**

**Batch: B5**

**Assignment: 7**

**Title of assignment: Implementation of AES – Advanced Encryption Standard**

**Title:**

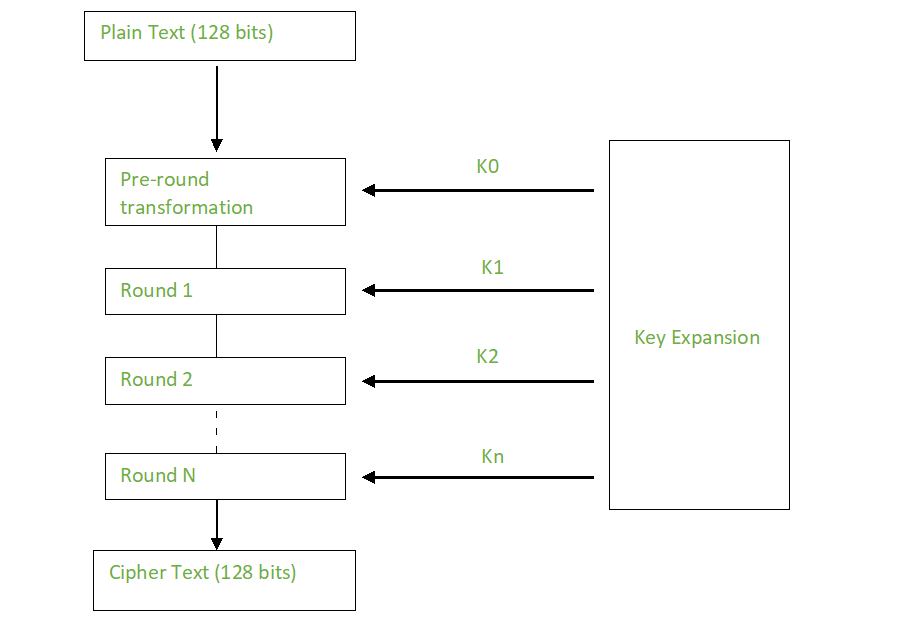
Implementation of Advanced Encryption Standard

**Aim:**

To develop and implement the Advanced Encryption Standard and to do encryption and decryption on the input plaintext

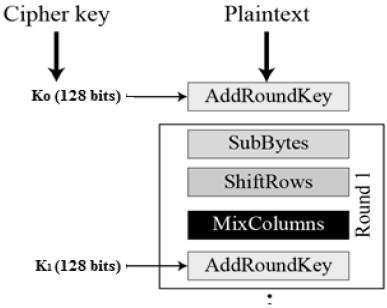
**Theory:**

* AES is an iterative rather than Feistel cipher. It is based on ‘substitution–permutation network’.
* Comprises of a series of linked operations, some of which involve replacing inputs by specific outputs (substitutions) and others involve shuffling bits around (permutations).
* AES performs all its computations on bytes rather than bits. Hence, AES treats the 128 bits of a plaintext block as 16 bytes. These 16 bytes are arranged in four columns and four rows for processing as a matrix
* the number of rounds in AES is variable and depends on the length of the key.
* AES uses 10 rounds for 128-bit keys, 12 rounds for 192-bit keys and 14 rounds for 256-bit keys. Each of these rounds uses a different 128-bit round key
* The features of AES are as follows
* Symmetric key symmetric block cipher
* 128-bit data, 128/192/256-bit keys
* Stronger and faster than Triple-DES
* Provide full specification and design details
* Software implementable in C and Java



**Encryption:**

A typical round of AES encryption comprises of four sub-processes. The first round process is depicted below –



**Byte Substitution (SubBytes)**

The 16 input bytes are substituted by looking up a fixed table (S-box) given in design. The result is in a matrix of four rows and four columns.

**Shiftrows**

Each of the four rows of the matrix is shifted to the left. Any entries that ‘fall off’ are re-inserted on the right side of row. Shift is carried out as follows −

1. First row is not shifted.
2. Second row is shifted one (byte) position to the left.
3. Third row is shifted two positions to the left.
4. Fourth row is shifted three positions to the left.

The result is a new matrix consisting of the same 16 bytes but shifted with respect to each other.

**MixColumns**

Each column of four bytes is now transformed using a special mathematical function. This function takes as input the four bytes of one column and outputs four completely new bytes, which replace the original column. The result is another new matrix consisting of 16 new bytes.

This step is not performed in the last round.

**Addroundkey**

The 16 bytes of the matrix are now considered as 128 bits and are XORed to the 128 bits of the round key. If this is the last round then the output is the ciphertext. Otherwise, the resulting 128 bits are interpreted as 16 bytes and we begin another similar round.

**Implementation of Advanced Encryption Standard**

**Code:**

import hashlib

from Crypto import Random

from Crypto.Cipher import AES

from base64 import b64encode, b64decode

class AESCipher(object):

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

self.block\_size = AES.block\_size

self.key = hashlib.sha256(key.encode()).digest()

def encrypt(self, plain\_text):

plain\_text = self.\_\_pad(plain\_text)

iv = Random.new().read(self.block\_size)

cipher = AES.new(self.key, AES.MODE\_CBC, iv)

encrypted\_text = cipher.encrypt(plain\_text.encode())

return b64encode(iv, encrypted\_text).decode("ütf-8")

def decrypt(self, encrypted\_text):

encrypted\_text = b64decode(encrypted\_text)

iv = encrypted\_text[:self.block\_size]

cipher = AES.new(self.key , AES.MODE.CBC, iv)

plain\_text = cipher.decrypt(encrypted\_text[self.block\_size:]).decode("utf-8")

return self.\_\_unpad(plain\_text)

def \_\_pad(self, plain\_text):

number\_of\_bytes\_to\_pad = self.block\_size - len(plain\_text) % self.block\_size

ascii\_string = chr(number\_of\_bytes\_to\_pad)

padding\_sgtr = number\_of\_bytes\_to\_pad \* ascii\_string

padded\_plain\_text = plain\_text + padding\_str

return padded\_plain\_text

@staticmethod

def \_\_unpad(plain\_text):

last\_character = plain\_text[len(plain\_text) - 1:]

return plain\_text[:ord(last\_character)]

key = input("Enter Key:")

obj = AESCipher(key)

str = input("Enter input: ")

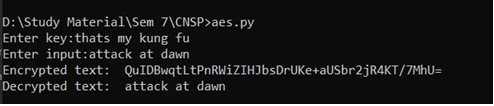
cipher = obj.encrypt(str)

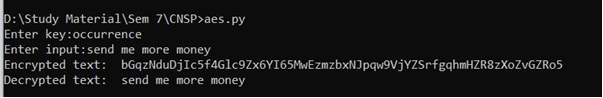
print(cipher)

plain\_text = obj.decrypt(cipher)

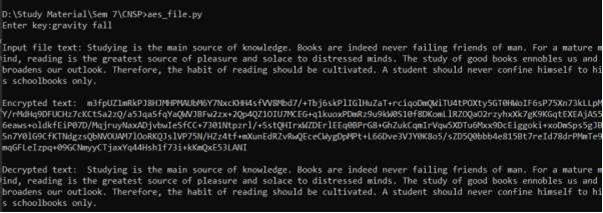
print(plain\_text)

**Output:**





**File Input:**



**Conclusion:**

Performed the experiment successfully. Encrypted the data

with the provided key. Output of this encryption is decrypted to match

the plaintext that was inputted by the user as shown in the above

diagram.