INTRODUCTION

What is encryption?

Encryption is the process of converting information into a secret code that conceals the real meaning of the information. Cryptography is the study of encrypting and decrypting information.Encryption has long been used to protect sensitive data by governments in the past.Encryption is used today to protect data stored on computers and storage devices, as well as data in transit across networks.

Unencrypted data is referred to as plaintext in computing, whereas encrypted data is referred to as ciphertext.Encryption algorithms, often known as ciphers, are the formulae used to encode and decode communications.A cipher's algorithm must include a variable in order to be effective.The variable, known as a key, is what distinguishes the output of a cipher.When an unauthorized party intercepts an encrypted communication, the intruder must determine the cipher the sender used to encrypt the message, as well as which keys were used as variables.The amount of time and complexity involved in predicting this information.

Encryption is critical in safeguarding a wide range of information technology (IT) assets.It includes the following features:

- **Confidentiality** encodes the message's content.
- **Authentication** confirms a message's origin.
- **Integrity** establishes that the contents of a communication have not been altered since it was transmitted.
- **Nonrepudiation** prohibits the sender from denying sending the encrypted communication.

AES Encryption technique

The AES Encryption algorithm is a 128-bit symmetric block cipher method. AES is currently one of the greatest encryption systems available, since it seamlessly mixes speed and security, allowing us to go about our regular online activities without interruption.

Given its advantages, it is hardly surprising that AES has become the industry standard for encryption.AES encryption keys are available in three lengths. Each key length has a unique number of key combinations:

- Key length of 128 bits: 3.4 x 1038
- Key length of 192 bits: 6.2 x 1057
- Key length of 256 bits: 1.1 x 1077

Although the length of the key in this encryption technique fluctuates, the block size - 128 bits (or 16 bytes) - remains constant.

BUT WHY ARE THERE DIFFERENT KEY LENGTH? And, since the 256-bit key is the most powerful (even referred to as "military-grade" encryption), why don't we just use it all the time?

It is all about the resources at our disposal.

For example, an app that utilizes AES-256 rather than AES-128 may deplete your phone's battery faster. Fortunately, modern technology has reduced the resource difference to the point where there is no reason not to employ 256-bit AES encryption.

WORKING OF AES ALGORITHM

→AES Data Units

AES refers to data in five different units: bits, bytes, words, blocks, and states. The bit is the smallest and most fundamental unit; all other units are represented in terms of smaller ones. Non-atomic data units include the byte, word, block, and state.

1.Bit

A bit in AES is a binary digit having the value 0 or 1.

2.Byte

A byte is an eight-bit group that can be handled as a single entity, a row matrix (1x8) of eight bits, or a column matrix (8x1).When the matrix is regarded as a row matrix, the bits are entered from left to right; when the matrix is treated as a column matrix, the bits are placed from top to bottom.

3.Word

A word is a set of 32 bits that can be handled as a single entity, a four-byte row matrix, or a four-byte column matrix. When it is handled as a row matrix, the bytes are placed from left to right; when it is treated as a column matrix, the bytes are entered from top to bottom.

4.Block

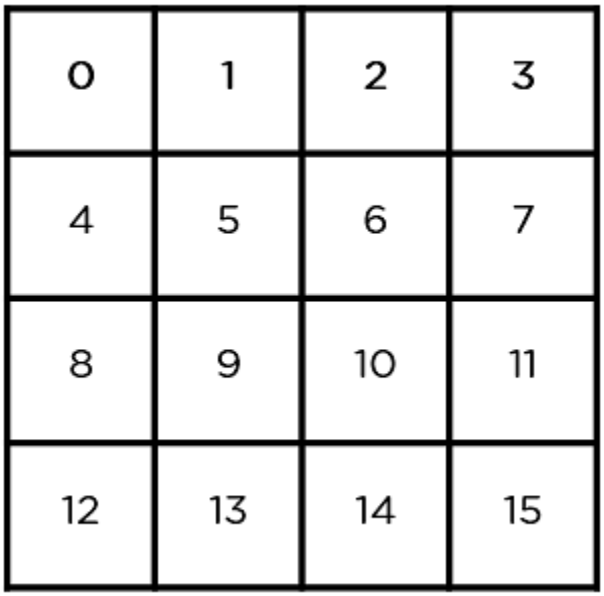
AES is used to encrypt and decode data blocks.In AES, a block is a collection of 128 bits. A block, on the other hand, can be represented as a 16-byte row matrix.

5.State

AES uses numerous rounds, each of which consists of several phases. The data block is moved from one stage to the next. AES utilizes the word data block at the beginning and conclusion of the encryption; before and after each stage, the data block is referred to as a state.

→Flow Of Algorithm

AES works by transmitting information between multiple steps. Since a single block is 16 bytes, a 4x4 matrix holds the data in a single block, with each cell holding a single byte of information.



A state array is the matrix seen in the figure above.Similarly, the original key is extended into (n+1) keys, where n is the number of rounds to be employed in the encryption process.For a 128-bit key, the number of rounds is 16, and the number of keys to be created is 10+1, for a total of 11 keys.

→Encryption Phase

The actions outlined above must be completed progressively for each block. It connects the constituent blocks to generate the final ciphertext after successfully encrypting them.The procedure is as follows:

```
def encrypt(pt):
    if chr(255) in pt:
        raise ValueError("Unsupported character")
    totalPadding = 16 - len(pt) % 16
    if totalPadding !=0:
        pt += chr(255)
    for i in range(totalPadding-1):
        pt += " "
    test = ""
    for i in range(0, len(pt), 16):
        curr_pt = pt[i : i + 16]

        state = block_to_state(curr_pt)

        for i in range(4):
            for j in range(4):
                state[i][j] = "%X" % ord(state[i][j])

        cstate = cipher(state, KEY)
        cstate = state_to_block(cstate)
        # chk = [chr(int(x, base=16)) for x in cstate]
        s = "".join(cstate)
        test = test + s
    return test
```

1. Dividing data into blocks

To begin, we must remember that AES is a block cipher.It encrypts data in blocks of bits rather than bit by bit like stream ciphers.Each of its blocks has a column of 16 bytes in a four-by-four arrangement.Because one byte has eight bits, the block size is 128 bits (16x8=128).As a result, the first step of AES encryption is to divide the plaintext into these blocks.

For example, it can be “Data Structures and Algorithms”.

Using the advanced encryption standard, the beginning of this phrase would become the following block :

D		U	R
A	S	C	E
T	T	T	S
A	R	U	

Note that this is only the first block of the text - the rest of the phrase would go into the next block and so on.

2. Key expansion

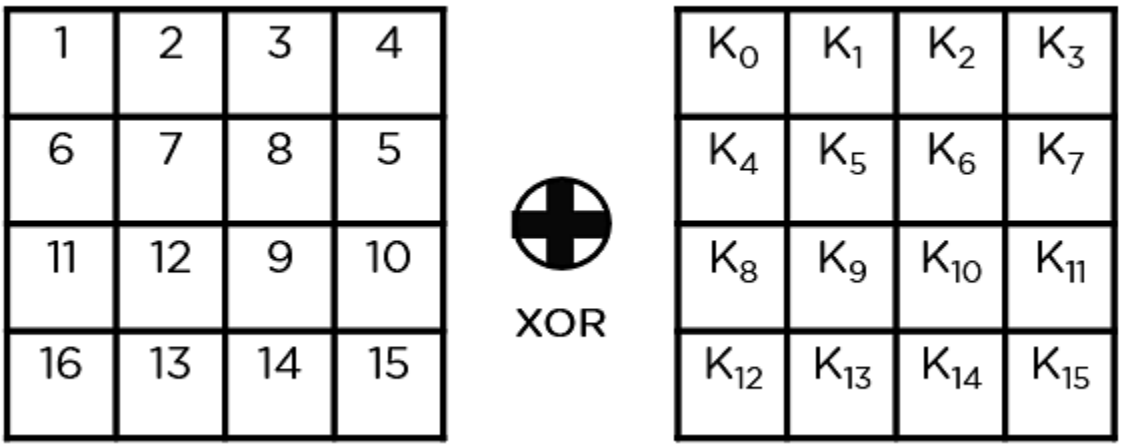
This is a critical stage in AES encryption. It uses Rijndael's key scheduling to generate fresh 128-bit round keys. The sentence will appear like a mess of random characters after applying Rijndael's key schedule. Rijndael's key schedule, on the other hand, employs predefined methods to encrypt each and every symbol, so these characters won't be that random after all.

```
'''
Performs key expansion to generate round keys
'''
def keyExpansion(key):
    RCon = ["01", "02", "04", "08", "10", "20", "40", "80", "1B", "36"]
    words = []
    for i in range(0, len(key), 4):
        wi = ""
        for x in key[i : i + 4]:
            wi += x
        words.append(wi)

    for i in range(4, 44):
        if i % 4 != 0:
            wi = "%X" % (int(words[i - 1], base=16) ^ int(words[i - 4], base=16))
            if len(wi) < 8:
                for i in range(8 - len(wi)):
                    wi = "0" + wi
            words.append(wi)
        else:
            t = int(SubWord(RotWord(words[i - 1])), base=16) ^ int(
                RCon[i // 4 - 1], base=16
            )
            wi = "%X" % (t ^ int(words[i - 4], base=16))
            if len(wi) < 8:
                for i in range(8 - len(wi)):
                    wi = "0" + wi
            words.append(wi)
    return words
```

3. Add Round Key

The block data, contained in the state array, is passed through an XOR function with the first key created (K0). It sends the produced state array to the following phase as input.



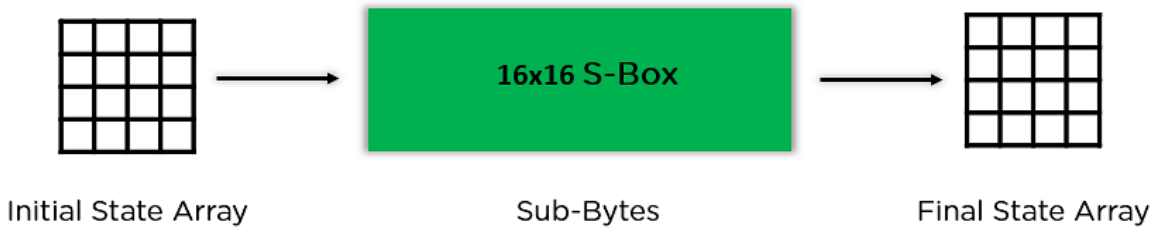

```python
def AddRoundKey(state, key):

    Nb = len(state)
    keys = [[None for j in range(4)] for i in range(Nb)]
    for j, k in enumerate(key):
        for i in range(0, 4):
            keys[i][j] = k[2 * i : 2 * i + 2]
    new_state = [[None for j in range(4)] for i in range(Nb)]

    for i in range(len(state)):
        for j in range(len(state[0])):
            # print('state',state[i][j], 'key',keys[i][j])
            new_state[i][j] = "%X" % (
                int(state[i][j], base=16) ^ int(keys[i][j], base=16)
            )
    return normalize_state(new_state)
```

4. Byte substitution

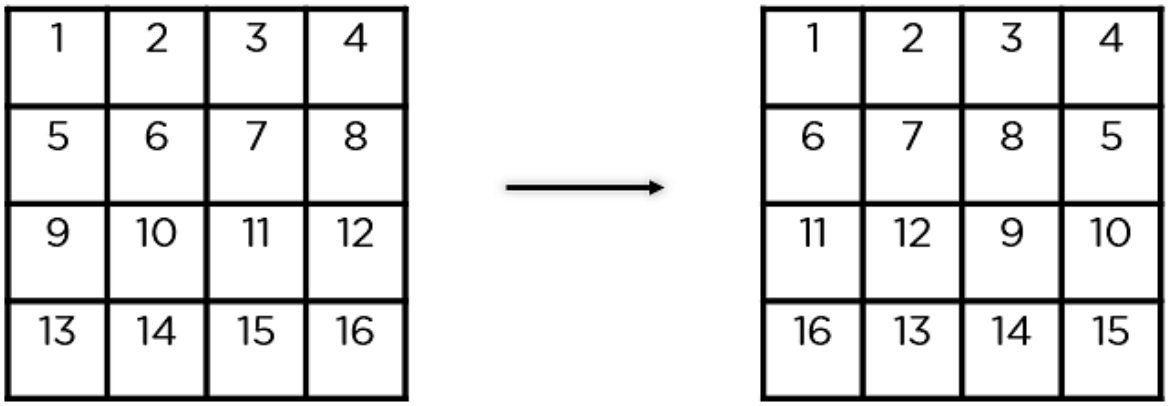
This phase divides each byte of the state array into two equal halves and transforms it to hexadecimal. These are the rows and columns that have been mapped with a substitution box (S-Box) to create new values for the final state array.




```
'''
Substitute bytes transformation
'''
def state_subBytes(state):
    state = normalize_state(state)
    subtest = [[0 for _ in range(4)] for _ in range(4)]
    for i in range(4):
        for j in range(4):
            subtest[i][j] = get_sub(state[i][j])
    return subtest
```

5. Rows Shifting

It switches the row items around. The first row is skipped. It moves the components in the second row to the left by one place. It also moves the items in the third row two places to the left and the last row three positions to the left.

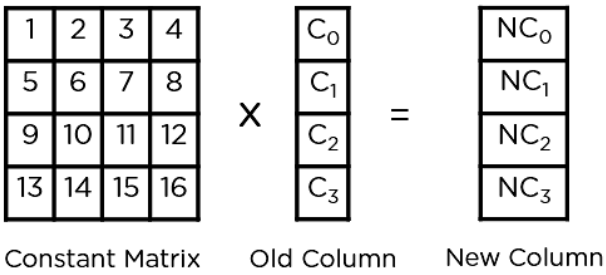



```
'''
Shift Rows Transformation
'''
def shiftRows(state):
    state = normalize_state(state)
    statec = state[:]
    for i in range(1, 4):
        temp = deque(statec[i])
        temp.rotate(-i)
        statec[i] = list(temp)
        # state[i] = state[i][i:]+state[i][:i]

    return statec
```

6. Mixing columns

It multiplies a constant matrix by each column in the state array to produce a new column for the next state array. After multiplying all of the columns by the same constant matrix, you receive your state array for the following step. This stage is not to be completed in the final round.

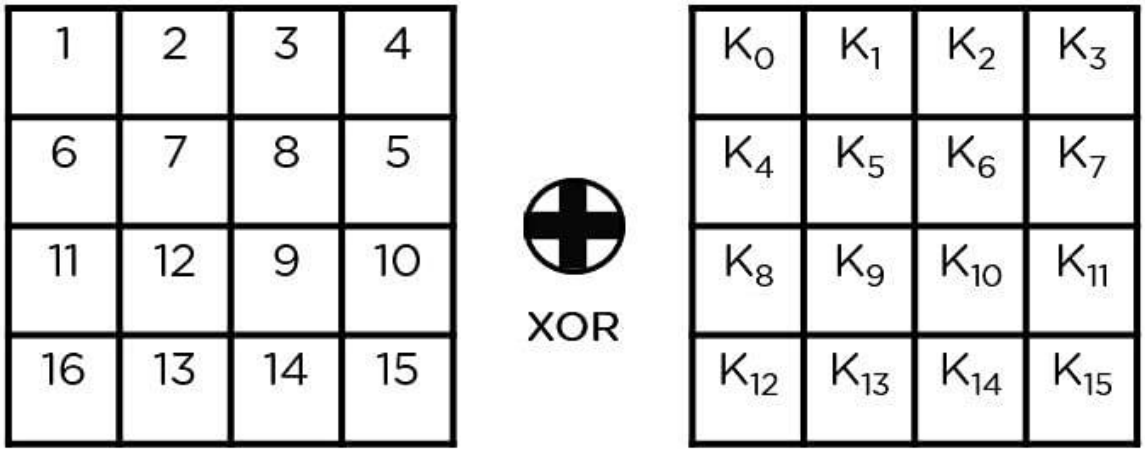



```python
'''
GF Field multiplication
'''
def gmul(a, b):
    p = 0
    for c in range(8):
        if b & 1:
            p ^= a
        a <<= 1
        if a & 0x100:
            a ^= 0x11B
        b >>= 1
    return p

'''
Single Column MixColumns
'''
def mixColumn(arr):
    a, b, c, d = arr
    v1 = gmul(a, 2) ^ gmul(b, 3) ^ gmul(c, 1) ^ gmul(d, 1)
    v2 = gmul(a, 1) ^ gmul(b, 2) ^ gmul(c, 3) ^ gmul(d, 1)
    v3 = gmul(a, 1) ^ gmul(b, 1) ^ gmul(c, 2) ^ gmul(d, 3)
    v4 = gmul(a, 3) ^ gmul(b, 1) ^ gmul(c, 1) ^ gmul(d, 2)
    return [v1, v2, v3, v4]
```

7. Adding round key

The round key we got in the *key expansion* section is added to the block we got in the previous section after the process of column mixing.




```python
def AddRoundKey(state, key):

    Nb = len(state)
    keys = [[None for j in range(4)] for i in range(Nb)]
    for j, k in enumerate(key):
        for i in range(0, 4):
            keys[i][j] = k[2 * i : 2 * i + 2]
    new_state = [[None for j in range(4)] for i in range(Nb)]

    for i in range(len(state)):
        for j in range(len(state[0])):
            # print('state',state[i][j], 'key',keys[i][j])
            new_state[i][j] = "%X" % (
                int(state[i][j], base=16) ^ int(keys[i][j], base=16)
            )
    return normalize_state(new_state)
```

8. Repeat, Repeat, Repeat

The AES encryption algorithm will now go through **many more rounds** of byte substitution, rearranging rows and columns, and adding a round key.The number of identical rounds that the data passes through is determined by the length of the AES key:

- 9 rounds of 128-bit encryption
- 11 rounds for 192-bit keys
- 13 rounds of 256-bit encryption

So, for example, in the case of 256-bit key encryption, the data runs through the previously indicated processes 13 times in a row.**However, that's still not the end of it.**

After the previously indicated 9, 11, or 13 rounds of encryption, there is one more round. During this extra round, the algorithm simply performs byte substitution, row shifts, and adds a round key. It omits the process of combining columns.Because that would be redundant at this time. In other words, this action would consume an inordinate amount of processing power while making no major changes to the data.As a result, at the end of the encryption process, the data will have gone through the following rounds:

- 10 cycles of a 128-bit key
- 12 cycles for a 192-bit key
- 14 rounds of 256-bit encryption

```
'''
Ciphers a given text with a given key
'''
def cipher(state, Key):
    global KEY

    Key = KEY

    roundkeys = keyExpansion(Key)
    round = 0
    newState = AddRoundKey(state, roundkeys[round * 4 : round * 4 + 4])

    # first 9 rounds have 4 transformations
    for round in range(1, 10):
        newState = state_subBytes(newState)
        newState = shiftRows(newState)
        newState = mixColumns(newState)
        newState = AddRoundKey(newState, roundkeys[round * 4 : round * 4 + 4])

    # Last round has only 3 transformations (skips mixColumns)
    newState = state_subBytes(newState)
    newState = shiftRows(newState)
    newState = AddRoundKey(newState, roundkeys[40:44])

    return newState
```

→Decryption Phase

The AES ciphertext may be recovered to its original state via **inverse encryption**.As previously stated, the advanced encryption standard employs symmetric cryptography,in other words, the same key is used for both data encryption and decryption.

In this sense, it differs from asymmetric encryption methods, which need both public and private keys.So, in our case, AES decryption starts with the inverted round key then reverses each process (shifting rows, byte replacement, and, subsequently, column mixing) until it deciphers the original message.

```
def decrypt(test):
    testc = ""
    for i in range(0, len(test), 32):

        curr_pt = test[i : i + 32]
        curr_pt = [curr_pt[i : i + 2] for i in range(0, len(curr_pt), 2

        state = block_to_state(curr_pt)

        cstate = decipher(state, KEY)
        cstate = state_to_block(cstate)
        chk = [chr(int(x, base=16)) for x in cstate]
        s = "".join(chk)
        testc = testc + s
    return testc.split(chr(255))[0]
```

```
'''
Deciphers a given text with a given key
'''
def decipher(state, Key):
    global KEY
    Key = KEY
    roundkeys = keyExpansion(Key)
    round = 10
    newState = AddRoundKey(state, roundkeys[round * 4 : round * 4 + 4])
    for round in range(9, -1, -1):
        newState = shiftRowsInv(newState)
        newState = state_subBytesInv(newState)
        newState = AddRoundKey(newState, roundkeys[round * 4 : round * 4 + 4])
        if round != 0:
            newState = invmixColumns(newState)
    return newState
```

COMPLEXITY ANALYSIS

Time Complexity

AES encryption has a time complexity of O(1), which means that the time required to encrypt a block of data is constant regardless of block size. This is because AES is a block cipher that encrypts and decrypts data using a fixed-size data block (128 bits) and a fixed-size key (128, 192, or 256 bits). AES's encryption and decryption operations are based on a predetermined set of changes that are performed to the data block in a precise order. As a result, the time required to encrypt or decode a block of data is independent of block size and is defined only by the number of transformations used.

When you place them in a mode of operation to encrypt lengthier messages, the complexity is generally O(m), where m is the message size, because you have O(m) blocks of data to encrypt.

Space Complexity

The space complexity of AES is O(1), which means that it is constant and does not rely on the amount of the data being encrypted. This is because AES encrypts and decrypts data using a fixed-size data block (128 bits) and a fixed-size key (128, 192, or 256 bits). AES's encryption and decryption operations are based on a predetermined set of changes that are performed to the data block in a precise order. As a result, the amount of memory needed to conduct these modifications is constant and independent of the size of the data to be encrypted.