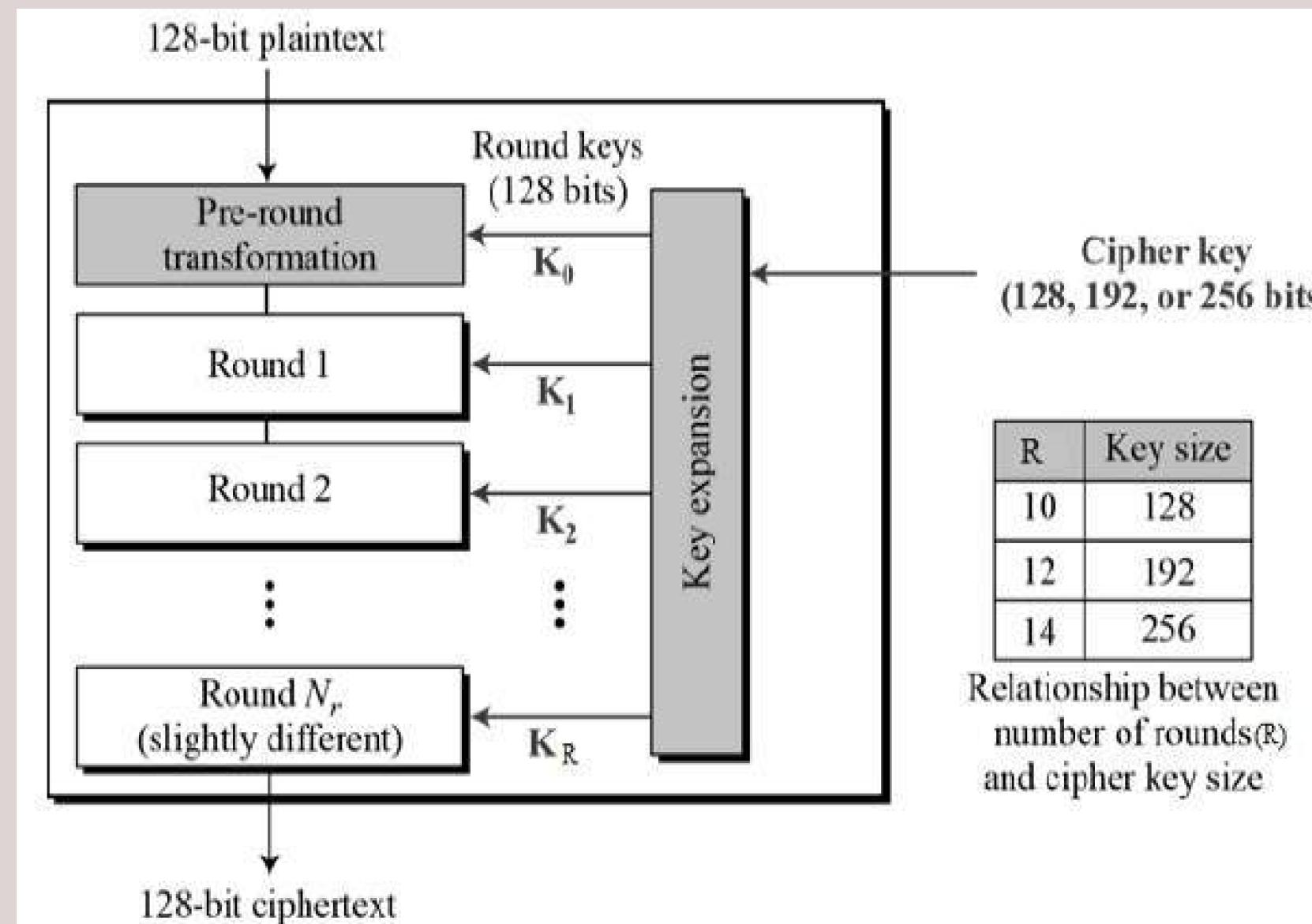


AES-Advanced Encryption Standard

Aditya Peer 2020355
Vedant Bothra 202060

AES-Advanced Encryption Standard



The AES system is a widely used symmetric key encryption algorithm that is used to secure sensitive data in various applications. It is a block cipher which operates on 128-bit blocks of data at a time, and can use keys of length 128, 192, or 256 bits. The number of rounds performed by the algo depends on the key length, it can perform 10, 12 or 14 rounds.

Our Project 1

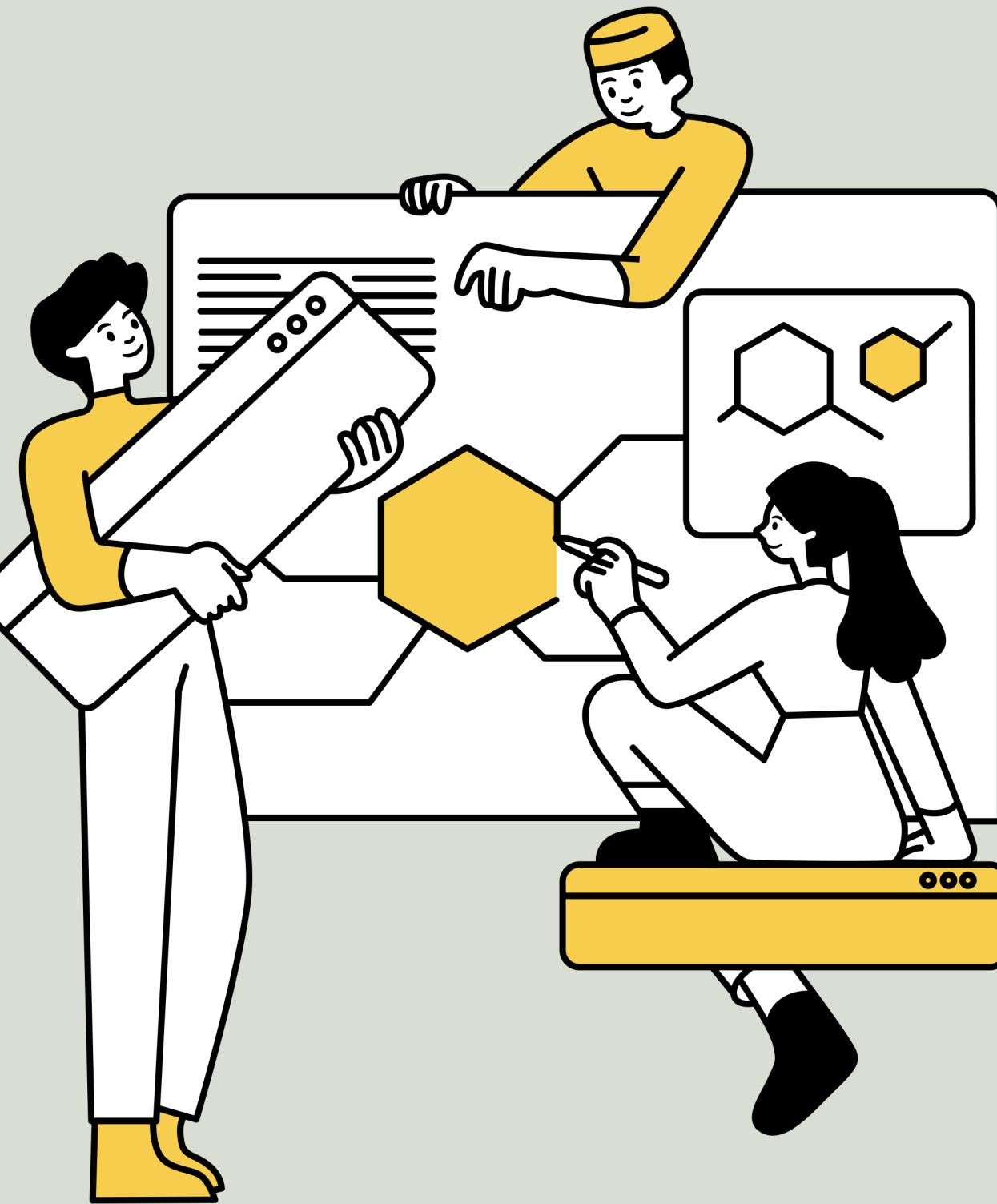
We have to implement project 1 which is to develop a program to encrypt (and similarly decrypt) a 128-bit plaintext using AES that uses keys of size 128 bit, and 10 rounds (repeat, 10 rounds).

The AES system we have implemented uses keys of size 128 bit, and 10 rounds.

To use the system we need a plain_text and a master key, these are hexadecimal strings of length 32, and are equal to 128 bits. In the system you either enter these strings on your own, or randomly generate plain_text and master key for convenience.

In the report we use,

```
plain_text = "dd06309f04da87df644726e304234012"  
master_key = "ddee540d70661c716d12c764c450ecee"
```



Key Expansion:

```
def start(bytes):
    ...
    #Generating random key of 16 Bytes, and then Expansion of the Keys from key [0,43] and
    generate 11 ley strings in the resultes array.
    ...
key=[]
for i in range(0,len(bytes)-1):
    if(i%2==0):
        key.append(bytes[i:i+2])
res = keyExpansion(key)
print("Master key: " + "".join(key))
return res
```

The AES key expansion algorithm takes as input a four-word (16-byte) key and produces a linear array of 44 words (176 bytes). This is sufficient to provide a four-word round key for the initial AddRoundKey stage and each of the 10 rounds of the algorithm.

Here in the **function start** we give the master key as input and convert this into a 4x4 bytes matrix , **then pass this matrix to KeyExpansion function.**

Key Expansion Function:

```
def keyExpansion(key):
    """
    Create a word array w, and to hold 44 tuples,First fill out 4 words based on the key, then fill out the rest based on previous words, rotword,
    subword and rcon values and if multiple of 4 use rot, sub, rcon etc and at last xor the hex values.
    """
    w = [()]*44

    for i in range(4):
        w[i] = (key[4*i], key[4*i+1], key[4*i+2], key[4*i+3])

    for i in range(4, 44):
        temp = w[i-1]
        word = w[i-4]

        if i % 4 == 0:
            rotateWord = RotationOfWord(temp)
            subword = SubstitutionOfWord(rotateWord)
            index = int(i/4)
            rconstant = tables.r_const_table[index]
            temp = HexandXor(subword, hex(rconstant)[2:])

            word = ''.join(word)
            temp = ''.join(temp)

            xorvalue = HexandXor(word, temp)
            w[i] = (xorvalue[:2], xorvalue[2:4], xorvalue[4:6], xorvalue[6:8])

    res = []
    st =""
    count =0
    for i in range(len(w)):
        st+=w[i][0] + w[i][1] + w[i][2] + w[i][3]
        count+=1
        if(count == 4):
            res.append(st)
            count =0
            st=""
    return res
```

Inside Key Expansion function we created an empty array of 44 size, then :

- The key is copied into the first four words of the expanded key.
 - The remainder of the expanded key is filled in four words at a time. Each added word $w[i]$ depends on the immediately preceding word, $w[i - 1]$, and the word four positions back, $w[i - 4]$.
 - In three out of four cases, a simple XOR is used. For a word whose position in the w array is a multiple of 4, a more complex function is used
- The Role of the complex g function:**

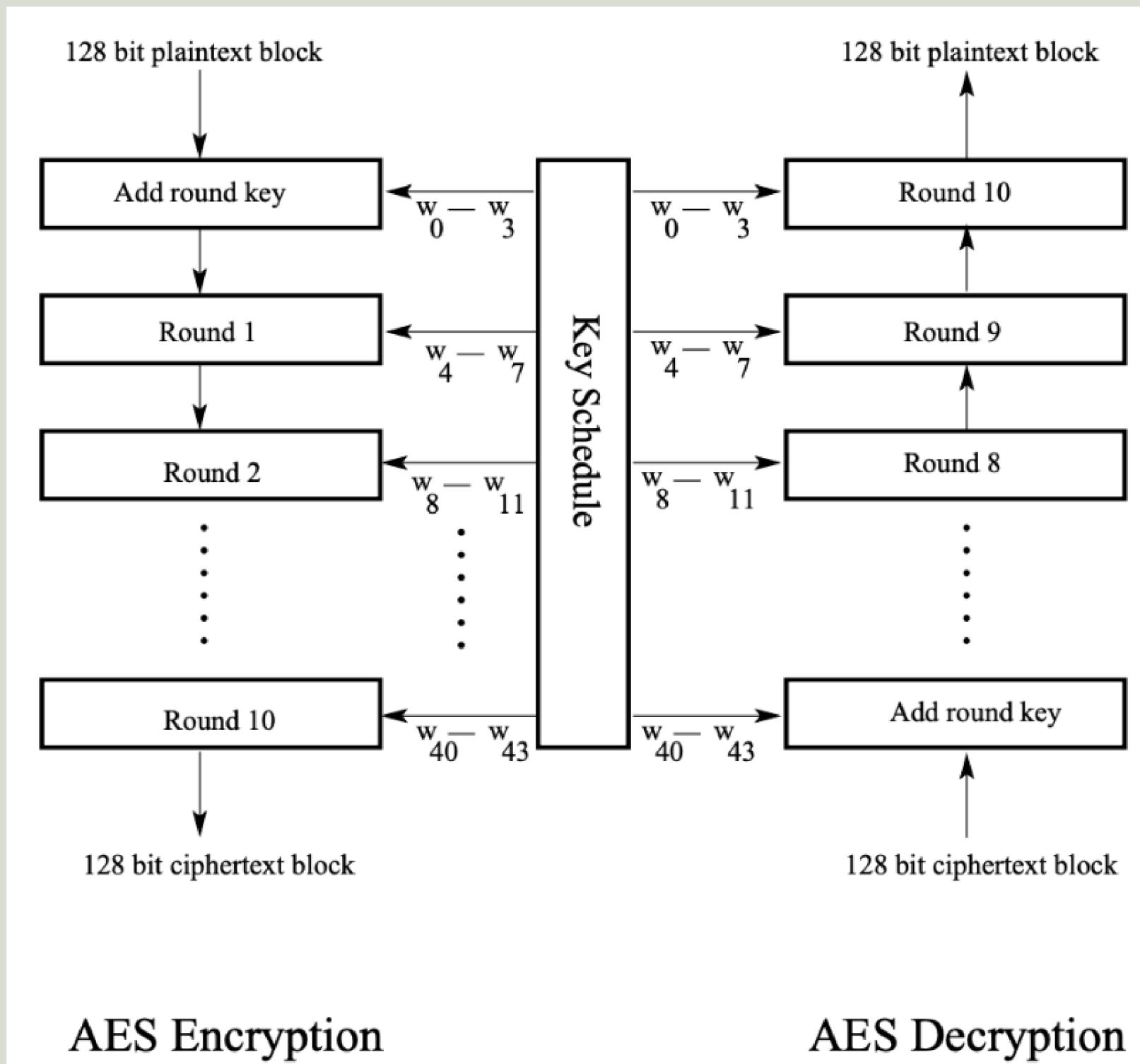
1. RotationofWord performs a one-byte circular left shift on a word.
This means that an input word $[B_0, B_1, B_2, B_3]$ is transformed into $[B_1, B_2, B_3, B_0]$.
2. SubstitutionofWord performs a byte substitution on each byte of its input word, using the S-box.
3. The result of steps 1 and 2 is XORed with a round constant, $Rcon[j]$.

The 11 Keys generated from the master key :

Key provided: ddee540d70661c716d12c764c450ecee

- 1 ddee540d70661c716d12c764c450ecee
- 2 8f207c11ff4660609254a70456044bea
- 3 7f93fba080d59bc012813cc44485772e
- 4 ec66cab6cb3517b7e326dbf3ab71a91
- 5 4dc44b3b21771a405f4577ff65f26d6e
- 6 d4f8d476f58fce36aacab9c9cf38d4a7
- 7 f3b088fc063f46caacf5ff0363cd2ba4
- 8 0e41c107087e87cda48b78cec746536a
- 9 d4acc3c1dcd2440c78593cc2bf1f6fa8
- 10 0f0401c9d3d645c5ab8f7907149016af
- 11 594378338a953df6211a44f1358a525e

Encryption:



Our AES system is based on this image.

The AES encryption algorithm consists of several key components that are used to encrypt the input data. These components include:

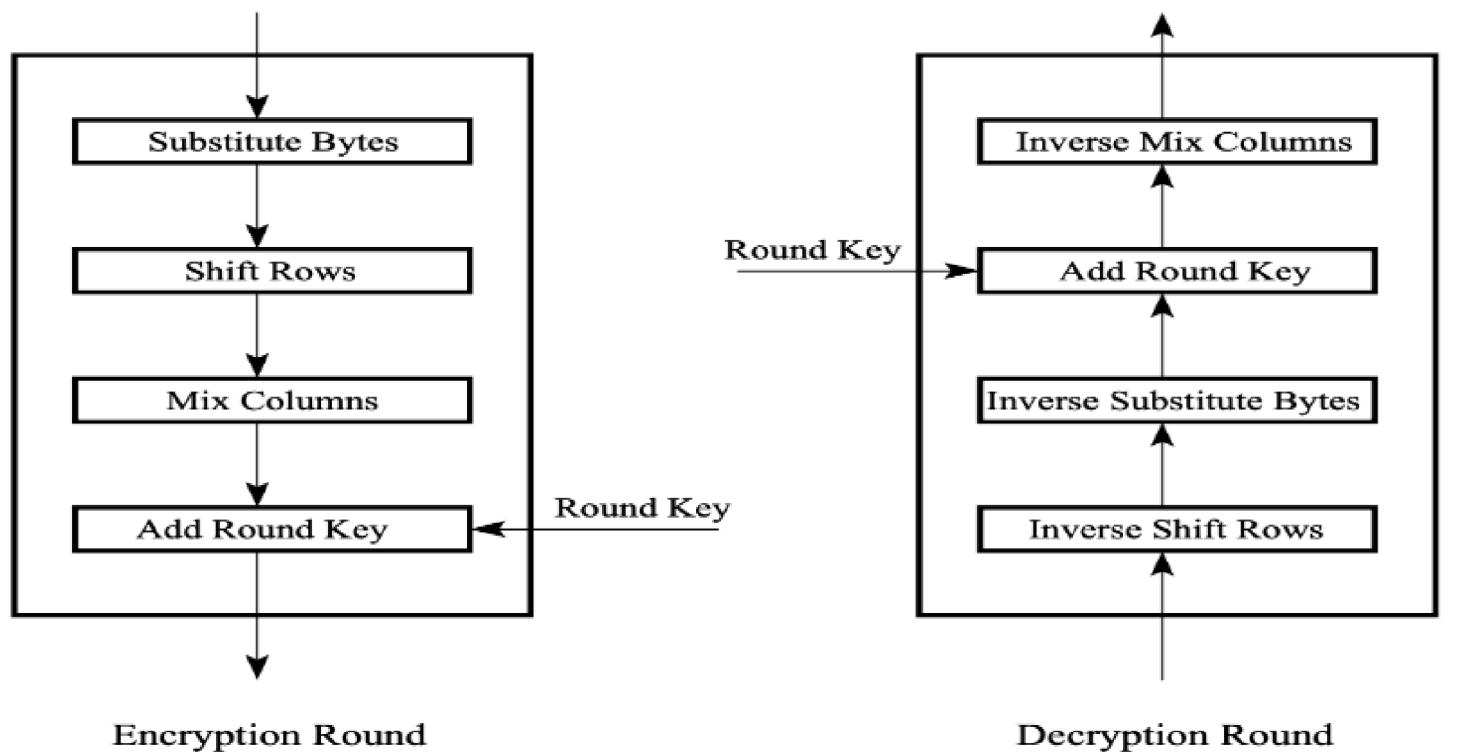
- **SubBytes Function.**
- **ShiftRows Function**
- **MixColumns Function**
- **AddRoundKey Function**

AES Encryption

AES Decryption

221	4	100	4
6	218	71	35
48	135	38	64
159	223	227	18

Round 1 to Round 9:



Each and every element of each of the 10 rounds of AES is programmed and no library is used.

In encrypt_128, the original plain_text is first converted to a state matrix, and is stored in [4]x[4] bytes (displayed in int) matrix. Where each column represents one word, this is done by the function get_state(). So the original plain_text gets converted to a 4 x 4 matrix.

Now this matrix is further worked upon, and is passed to Round 0 (Initial Round):

- During the initial round, the input plaintext matrix is XORed with the initial round key (this is the first key of the 11 keys generated by key expansion).
 - The **rounds 1 to 9** of the AES encryption algorithm consist of four steps that are applied in sequence given in the image. Each round uses a different key, round 1 uses the 2nd key generated by key expansion and so on.

===== ENCRYPTION BEGINS =====

0 00e8649274bc9bae0955e187c073acf
1 ae2a5a5201d0e63ebd7198188cb87b9e
2 7156feb7a79706c65e4880e79b141da8
3 3d403d9f2f15ec20d4e7656521ffc640
4 ac8fb454c34d9ef265ffa5631c19c92f
5 c0ab95116ed137cf5331a14e89a43b8d
6 b17d8f9f118a5c4a167e8a432534b0f0
7 f57b044725e855e9a7bfa0dbf35829aa
8 f9f08ac8bfa9d9bb92c2d78283f6700b
9 6d192af2158fe962672cad257e33291f
10 6530edf3d3e4987fa4d9a15bc65e4c61

Round 10 (Final Round):

This differs from other rounds as it does not have the mixcolumns step, the output of this round gives the cipher text.

Image shows the output of each round in hex string, round 10 gives the cipher text and round 0 is the initial round. This was done using encrypt_128(plain_text, master_key):

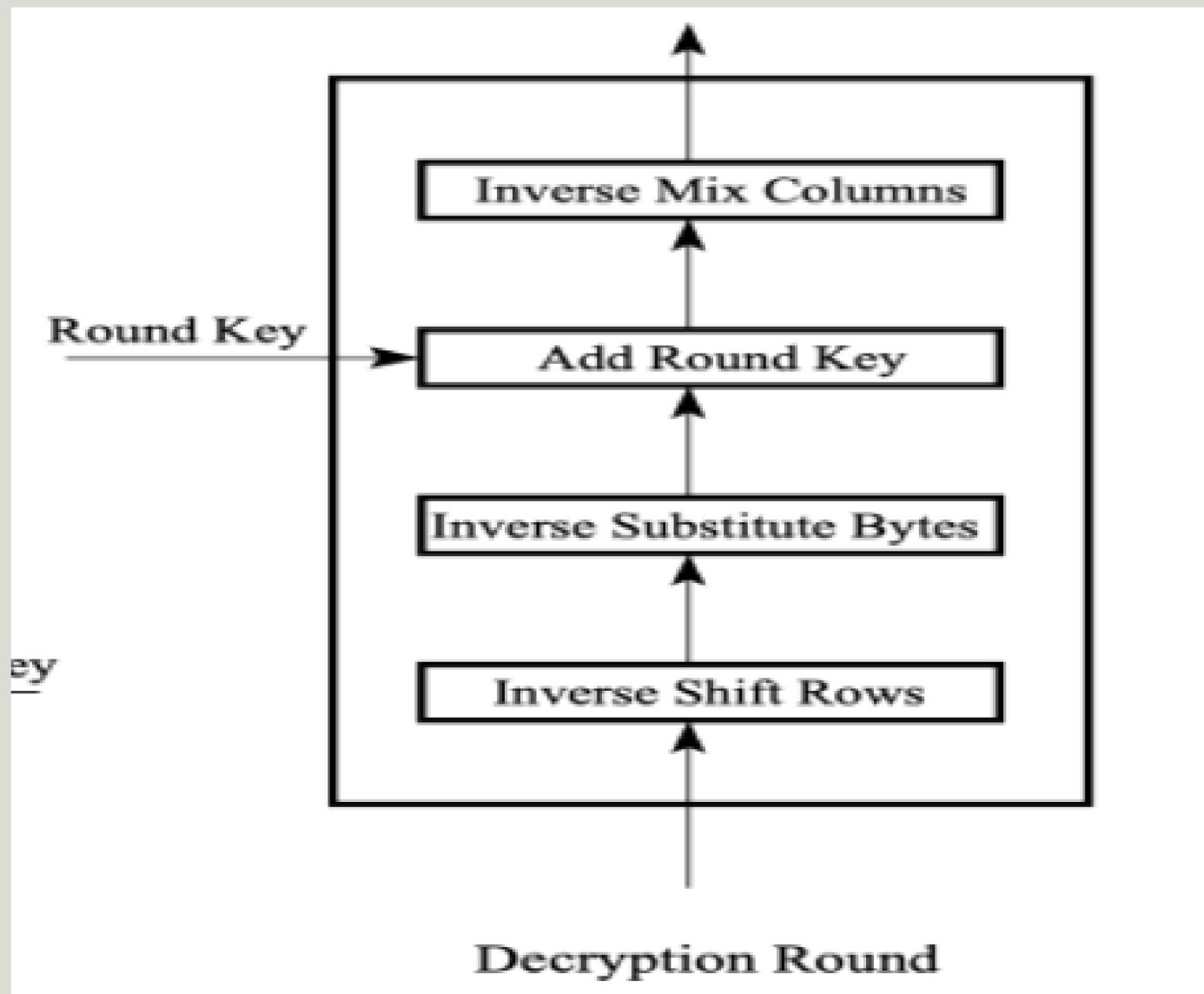
plain_text: dd06309f04da87df644726e304234012
master key: ddee540d70661c716d12c764c450ecee

The Cipher text produced at **10th round :**

cipher_text = 6530edf3d3e4987fa4d9a15bc65e4c61

Decryption:

This is done using the function `decrypt_128(cipher_text, master_key)`. The AES algorithm consists of several key components that are used to decrypt the input data. These components include:

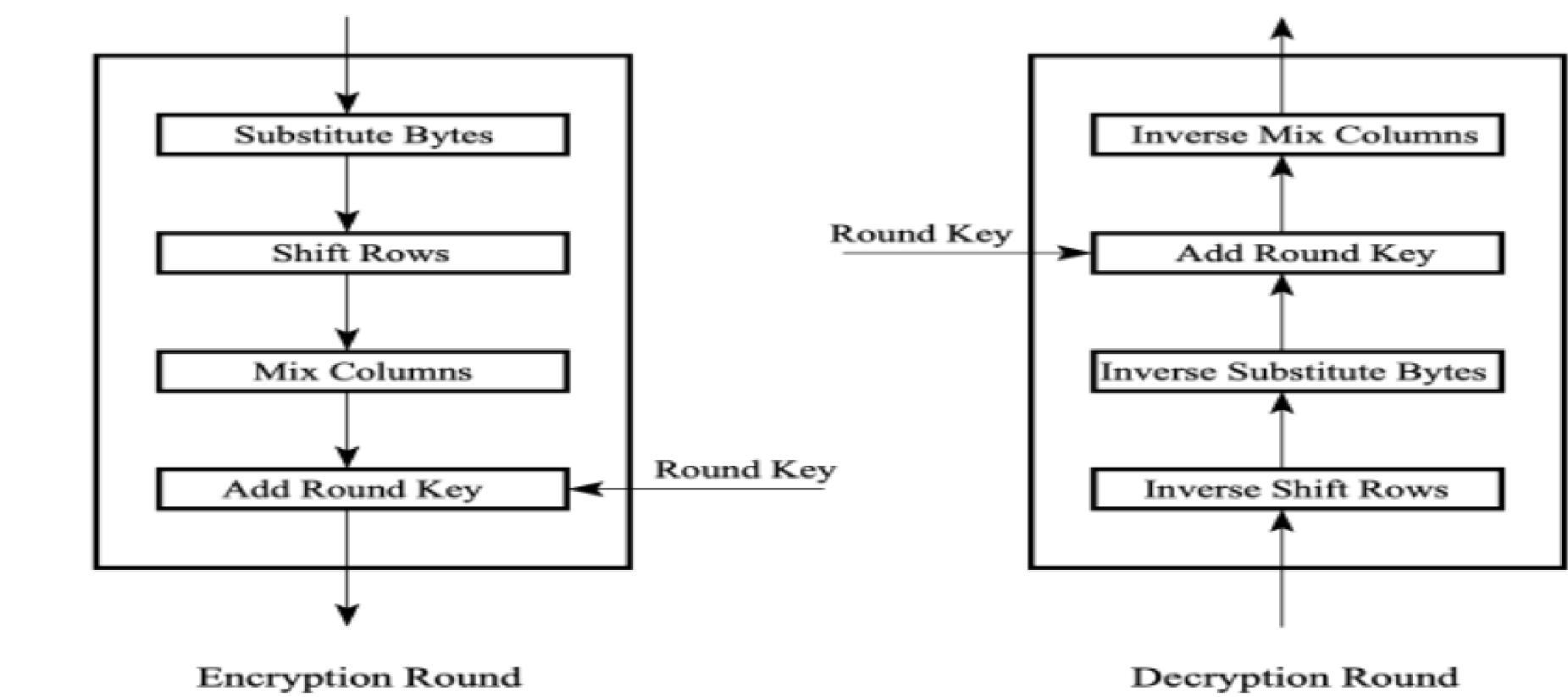


- **Inverse Shift Rows**
- **Inverse Substitute Bytes**
- **Add Round Key**
- **Inverse Mix Columns**

Similar to `encrypt_128`, the original `cipher_text` is first converted to a state matrix, and is stored in $[4] \times [4]$ bytes (displayed in int) matrix. Where each column represents one word, this is done by the function `get_state()`. So the original `cipher_text` gets converted to a 4×4 matrix.

Now this matrix is further worked upon, and is passed to **Round 0** (Initial Round). During the initial round, the input `cipher_text` matrix is XORed with the last round key (this is the 11th key of the 11 keys generated by key expansion).

Round 1 to Round 9:



The rounds 1 to 9 of the AES decryption algorithm consist of four steps that are applied in sequence given in the image. The steps used here are different in order and are inverse of the functions used in encryption.

Each of these rounds use a different key (in reverse order of what was used in encrypt_128). Round 9 uses the 2nd key generated by key expansion and so on.

Round 10 (Final Round):

===== DECRYPTION BEGINS =====

```
0 3c7395c05971a58985c3e5aaaf3d41e3f
1 99d30e2b082551e84f427eeaec8c3513
2 e69be0ac3f08a5a05c6af21e0d21fc9
3 c87e7e8c82f3e7db471873d63fff4a1a
4 ba3e325d9fc7e282ed492a8aa7629a2f
5 91e306152e16dd204dd48d899c730bf9
6 27594d091594b4db481627b7fd09ce4d
7 a388cdc25c52a4a958fabbb414b16f94
8 e470460b7ca321007a6cbef264e58ead
9 6365f8b092fc914f018f43e4ba9b1417
10 dd06309f04da87df644726e304234012
```

This differs from other rounds as it does not have the mixcolumns step but the order of the functions is the same, the output of this round gives the decrypted text.

Image shows the output of each round in hex string, round 10 gives the decrypted text and round 0 is the initial round. This was done using decrypt_128(cipher_text, master_key)

cipher_text: 6530edf3d3e4987fa4d9a15bc65e4c61
master key: ddee540d70661c716d12c764c450ecee

The decrypted text produced at 10th round is :

decrypted_text: dd06309f04da87df644726e304234012

a)

- decrypted_text matches the original plain_text which is **dd06309f04da87df644726e304234012**.
the ciphertext when decrypted yields the original plaintext.

Now as given in the assignment :

- b) Verify that the output of the 1st encryption round is same as the output of the 9th decryption round.
- c) Verify that the output of the 9th encryption round is same as the output of the 1st decryption round.

output of the 1st encryption round: ae2a5a5201d0e63ebd7198188cb87b9e
output of the 9th encryption round: 6d192af2158fe962672cad257e33291f

output of the 1st decryption round: 6d192af2158fe962672cad257e33291f
output of the 9th decryption round: ae2a5a5201d0e63ebd7198188cb87b9e

These screenshots are taken from the aes system we implemented, through them we can interpret that:

- The output of the 1st encryption round is same as the output of the 9th decryption round.
- The output of the 9th encryption round is same as the output of the 1st decryption round.

Testing AES with two pairs of 128 bit <plaintext, ciphertext>

1)

```
plain_text 1afe096fa27385cc1d851b64f9e4263b  
master key f960c3b4faef4eb7593853b1e8ba742e
```

==== ENCRYPTION BEGINS ====

```
0 e39ecadb589ccb7b44bd48d5115e5215  
1 5ceb7222153a7c352433276a59292988  
2 d23c298ef958eea7c2efd6b67471fb82  
3 788338ee343fddd5b005f96849fdce72  
4 5dc0c826270d408ad9fd34cc1fc938c1  
5 83993a5b9fc7cb4b642c729f2a36df5c  
6 7cc3fe7a6dd6130eaaeb95d10a9b2fa6  
7 91dfdf6dd1d5eb0357ee14693464f03  
8 4f76d520f5e4ea9d923a494e46cb3276  
9 81de64ff9a1317189250d1707a6f4d75  
10 513353e6a7217b9220a0be289bed46ae
```

```
output of the 1st encryption round: 5ceb7222153a7c352433276a59292988  
output of the 9th encryption round: 81de64ff9a1317189250d1707a6f4d75
```

==== DECRYPTION BEGINS ====

```
0 0c7d3e9db853e3164fa843adda1df051  
1 84693b38e68023b74f1f035e5a38872f  
2 81a4f87bc1f3848e965a9ee7dc9e585a  
3 10f62a243ce915daac14bbab672e7d3e  
4 ecc6404adb719e39430580b3e5ee1fdb  
5 4cd71878cc5407f735dde87ec0ba094b  
6 bc759940186b8b28e75407033becc145  
7 b56af61399df0f1925a3a55c92eb284e  
8 4a80ccc459c3a59336a54096cbe91002  
9 11de52596a7a00b91b587421820b1f03  
10 1afe096fa27385cc1d851b64f9e4263b
```

```
output of the 1st decryption round: 81de64ff9a1317189250d1707a6f4d75  
output of the 9th decryption round: 5ceb7222153a7c352433276a59292988
```

```
plain_text: 1afe096fa27385cc1d851b64f9e4263b  
cipher_text: 513353e6a7217b9220a0be289bed46ae  
decrypted_text: 1afe096fa27385cc1d851b64f9e4263b  
True
```

2)

```
plain_text 45eb6e86d07505516e41eae34dc54217  
master key cd3189ab009c0df2ed1022b0d8f68a1c
```

==== ENCRYPTION BEGINS ====

```
0 88dae72dd0e908a38351c8539533c80b  
1 fcbf793d36e1013a601f38e4cb0c814c  
2 5cd30a46eed69b9563ced7a87f6ff183  
3 34ed4aecfa52fef7635e1c0caf009b05  
4 6ddcd43b15aaacf4ab9cd0ba79b345ce  
5 84d08d0bd2dfa5524c06c1f97a6c9776  
6 0e92ef70bfe8f9a4eb9c12b84dca5eb7  
7 696791af4e3508bff9e3af542e1335e0  
8 955e1d7ae90ca7db76fb8cce60ead9e4  
9 eace332e039abf706f74e45792a20375  
10 8eab0dc2b39eaea9f021ce9c013db081
```

```
output of the 1st encryption round: fcbf793d36e1013a601f38e4cb0c814c  
output of the 9th encryption round: eace332e039abf706f74e45792a20375
```

==== DECRYPTION BEGINS ====

```
0 87b8699d7b927b31a83ac3514f8b085b  
1 2afe64691e0f35da3887a4b9d0585c8b  
2 f99679e12f119679997d810831853020  
3 ab9bc9a908de5851e974df49e34f996c  
4 5f9e7838b56f882b29505d00da700699  
5 3cac708b59de6ee2626d48fb68691f4  
6 18009c6b2d5814cef63d6687955bbfe  
7 4af60eec288ba15afba8672ad26614c2  
8 b0f8072905c00c27d0feb6801f087c69  
9 c41ee82b70d1e8d8ecc3940a2a5730ed  
10 45eb6e86d07505516e41eae34dc54217
```

```
output of the 1st decryption round: eace332e039abf706f74e45792a20375  
output of the 9th decryption round: fcbf793d36e1013a601f38e4cb0c814c
```

```
plain_text: 45eb6e86d07505516e41eae34dc54217  
cipher_text: 8eab0dc2b39eaea9f021ce9c013db081  
decrypted_text: 45eb6e86d07505516e41eae34dc54217  
True
```

Thank You!