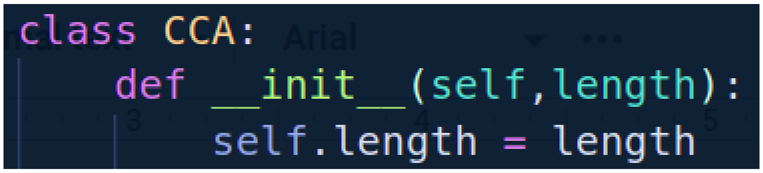
CCA ENCRYPTION SCHEME CODE DOCUMENTATION

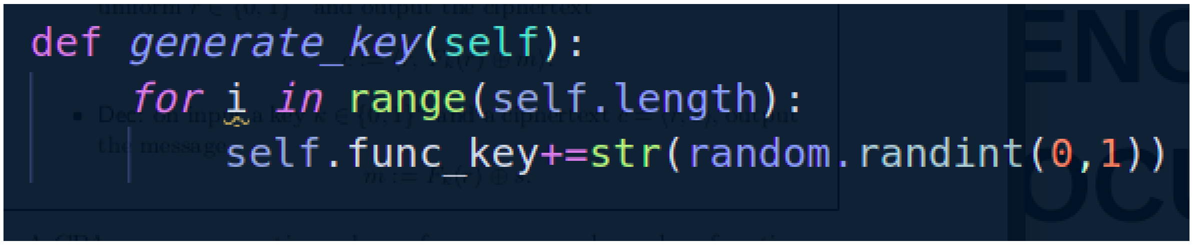
A Class named **CCA** is created incorporating all required variables and functions.



```
class CCA:
    def __init__(self,length):
        self.length = length
```

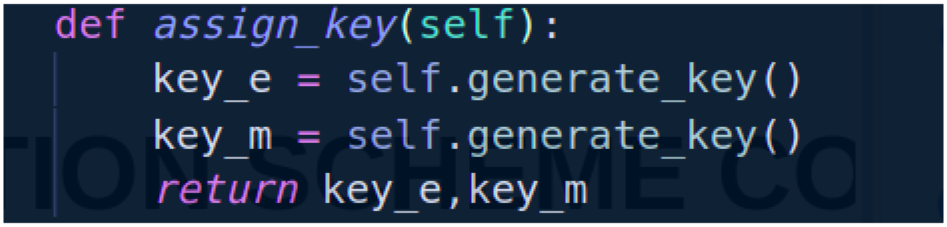
In init all the parameters passed to the class PRF are initialized,the parameters are as follows:

length : This variable stores the value of the length of the message(in bits representation)



```
def generate_key(self):
    for i in range(self.length):
        self.func_key+=str(random.randint(0,1))
```

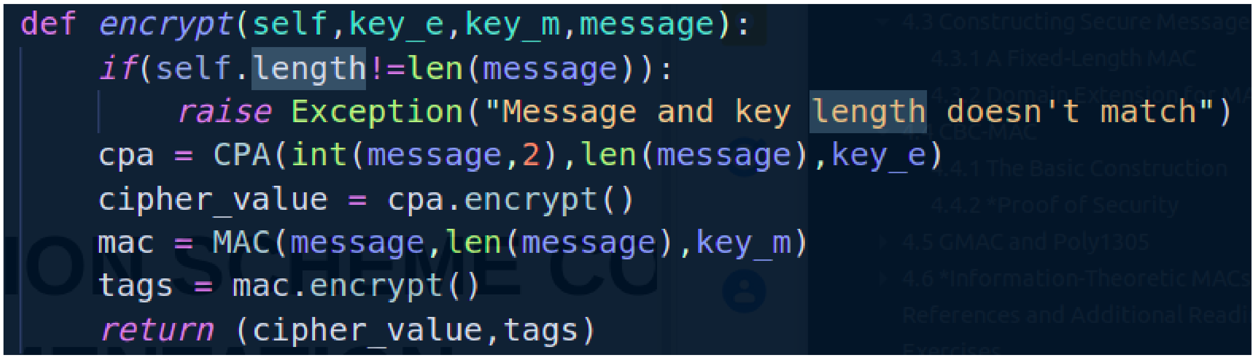
generate_key :
This function is used to generate an n bit uniform random string, it is used to generate the key.



```
def assign_key(self):
    key_e = self.generate_key()
    key_m = self.generate_key()
    return key_e,key_m
```

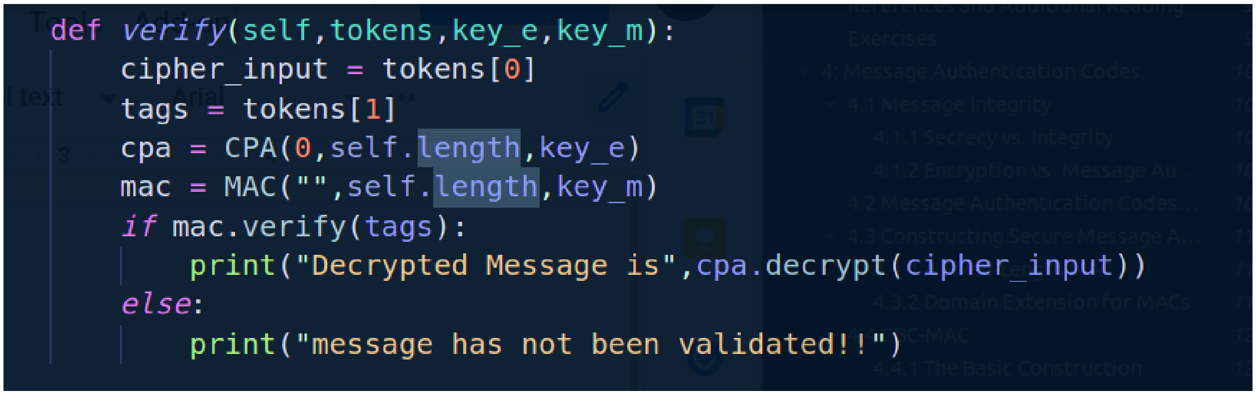
assign_key :

This function is used to initialize the key for encryption **key_e** and the key for MAC denoted by **key_m** respectively. To do so it uses generate_key function.



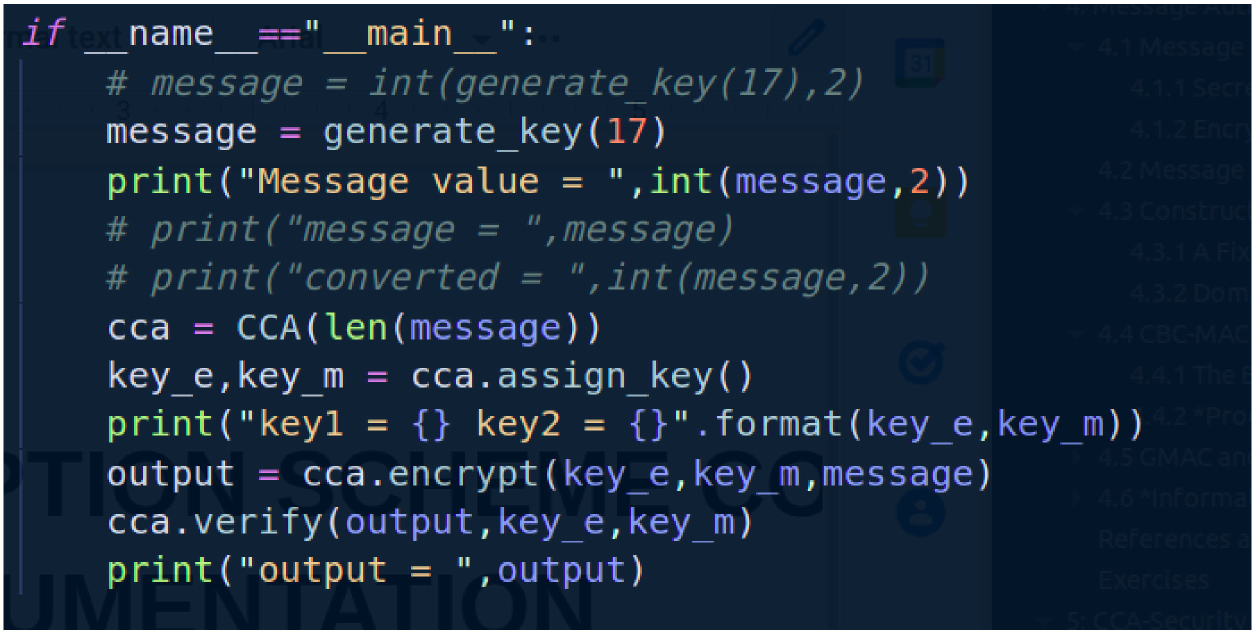
```
def encrypt(self,key_e,key_m,message):
    if(self.length!=len(message)):
        raise Exception("Message and key length doesn't match")
    cpa = CPA(int(message,2),len(message),key_e)
    cipher_value = cpa.encrypt()
    mac = MAC(message,len(message),key_m)
    tags = mac.encrypt()
    return (cipher_value,tags)
```

encrypt function takes key_e , key_m and message as input it encrypts the message using cpa algorithm implemented earlier and generates a cipher value. Now MAC function takes key_m, message and output message tags using the **fixed length** MAC function that was implemented, this function finally returns a tuple consisting of cipher value and tags.



```
def verify(self,tokens,key_e,key_m):
    cipher_input = tokens[0]
    tags = tokens[1]
    cpa = CPA(0,self.length,key_e)
    mac = MAC("",self.length,key_m)
    if mac.verify(tags):
        print("Decrypted Message is",cpa.decrypt(cipher_input))
    else:
        print("message has not been validated!!")
```

Verify function takes tags,cipher value along with **key_e** and **key_m**, it then creates cpa and mac schemes using the respective classes and using the corresponding keys **key_e** and **key_s**.
Now at first mac is used to verify the tags and once this tag is verified, cpa is used to decrypt the message.



```
if __name__=="__main__":
    # message = int(generate_key(17),2)
    message = generate_key(17)
    print("Message value = ",int(message,2))
    # print("message = ",message)
    # print("converted = ",int(message,2))
    cca = CCA(len(message))
    key_e,key_m = cca.assign_key()
    print("key1 = {} key2 = {}".format(key_e,key_m))
    output = cca.encrypt(key_e,key_m,message)
    cca.verify(output,key_e,key_m)
    print("output = ",output)
```

The main function takes all the required inputs and all required classes and keys are present.