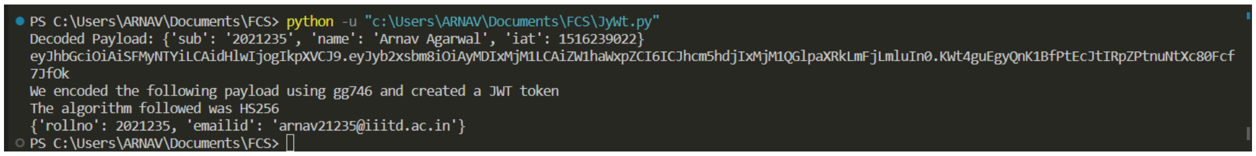
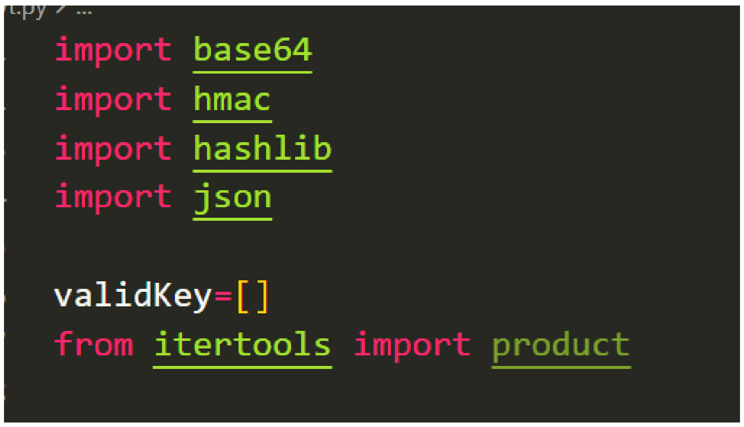
FCS Question 3

Final Output:



```
PS C:\Users\ARNAV\Documents\FCS> python -u "c:\Users\ARNAV\Documents\FCS\JyWt.py"
Decoded Payload: {'sub': '2021235', 'name': 'Arnav Agarwal', 'iat': 1516239022}
eyJhbGciOiAiSFMyNTYiLCAidHlwIjogIkpXVCJ9.eyJyb2xsbm8iOiAyMDIxMjM1LCAiZW1haWxpZCI6ICJhcm5hdjIxMjM1QGlpaXRkLmFjLmluIn0.KWt4guEgyQnK1BfPtEcJtIRpZPtnuNtXc80Fcf
7JfOk
We encoded the following payload using gg746 and created a JWT token
The algorithm followed was HS256
{'rollno': 2021235, 'emailid': 'arnav21235@iiitd.ac.in'}
PS C:\Users\ARNAV\Documents\FCS> 
```

Libraries used for this question are listed below:



```python
import base64
import hmac
import hashlib
import json

validKey=[]
from itertools import product
```

1)Token generated: token =

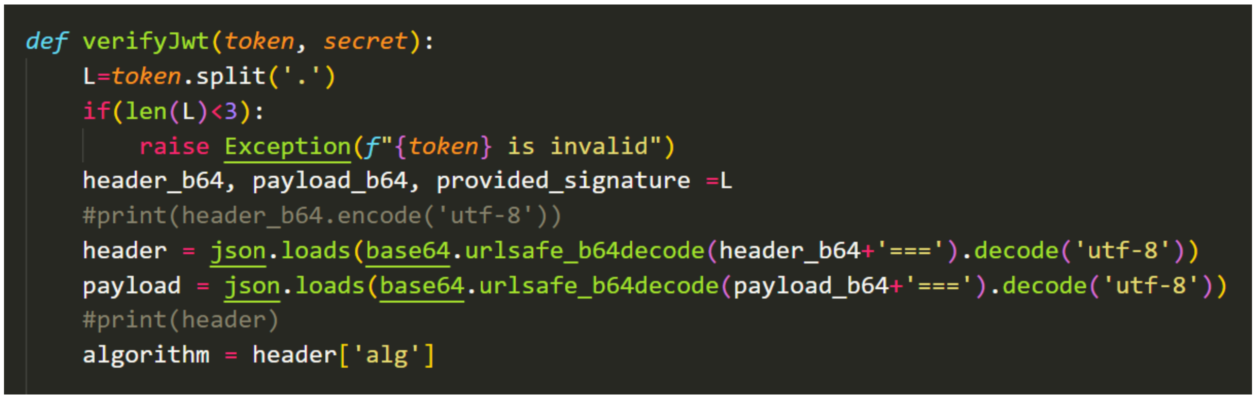
"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIyMDIxMjM1IiwibmFtZSI6IkFybmF2IEFnYXJ3YWwiLCJpYXQiOjE1MTYyMzkwMjJ9.OXNIctlVP0iQTKJf-stDW2fNo4KgEQ-_ZUDkt180eRk"

Secret: "adwidwiijdwiajdpppiwjdpwaljlsjdowijkkjksjwswdjawp"

The verifyJWT function takes two arguments the token and the secret

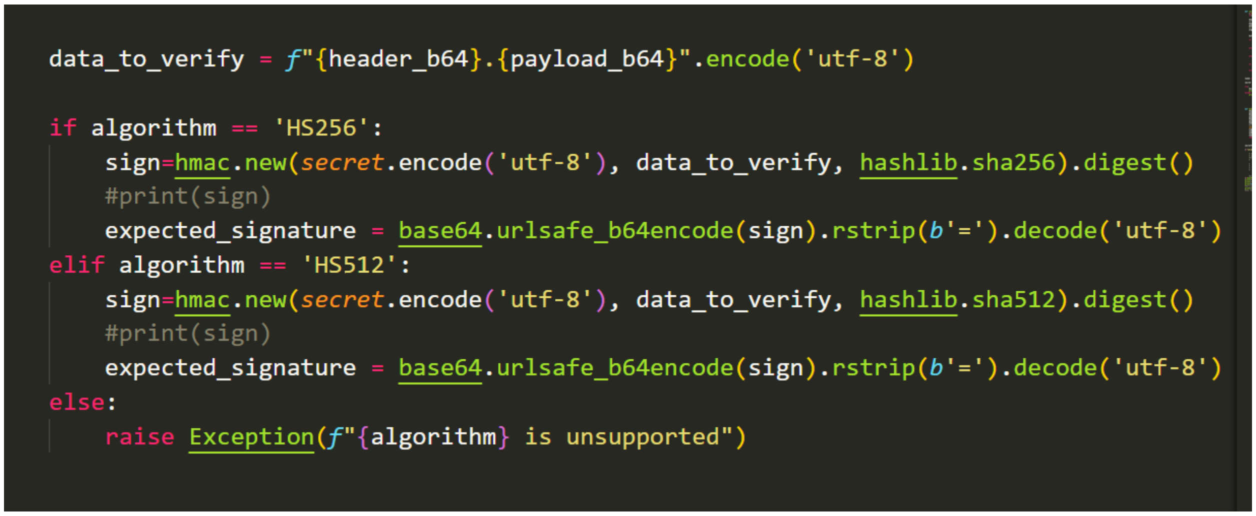
Step1: Check if the token is in the correct format. The header,payload and secret must be part of the token and separated by the . In case this is not true we raise an exception.

Step2: We extract the header and payload using the split function and decode the header and payload using base64.urlsafe_decode() .



```python
def verifyJwt(token, secret):
    L=token.split('.')
    if(len(L)<3):
        raise Exception(f"{token} is invalid")
    header_b64, payload_b64, provided_signature =L
    #print(header_b64.encode('utf-8'))
    header = json.loads(base64.urlsafe_b64decode(header_b64+'===').decode('utf-8'))
    payload = json.loads(base64.urlsafe_b64decode(payload_b64+'===').decode('utf-8'))
    #print(header)
    algorithm = header['alg']
```

Step3: Check the algorithm in the header. We have support for HS256 and HS512 so raise an exception if the algorithm is neither of these.If the algorithm is valid we create the expected signature using the secret(in bytes form),the combination of header and payload and the hashing algorithm(HS256 or HS512).



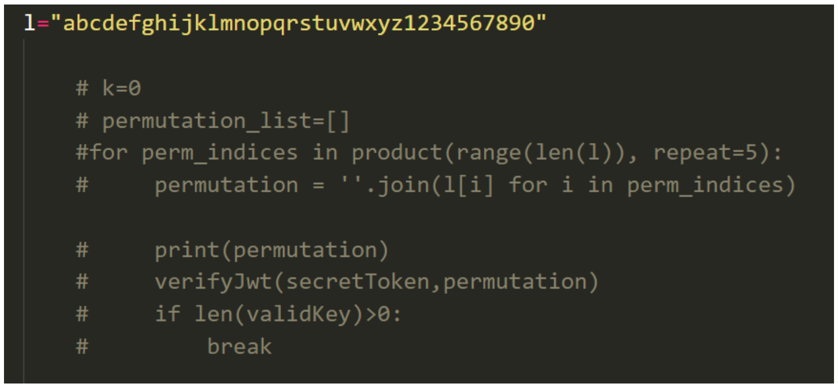
```python
data_to_verify = f"{header_b64}.{payload_b64}".encode('utf-8')

if algorithm == 'HS256':
    sign=hmac.new(secret.encode('utf-8'), data_to_verify, hashlib.sha256).digest()
    #print(sign)
    expected_signature = base64.urlsafe_b64encode(sign).rstrip(b'=').decode('utf-8')
elif algorithm == 'HS512':
    sign=hmac.new(secret.encode('utf-8'), data_to_verify, hashlib.sha512).digest()
    #print(sign)
    expected_signature = base64.urlsafe_b64encode(sign).rstrip(b'=').decode('utf-8')
else:
    raise Exception(f"{algorithm} is unsupported")
```

Step4: If the expected and provided signature is the same then the payload has not been tampered with and return payload. Else return that secret is either invalid or the token has been tampered with.

2)Token:eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiJmY3MtYXNzaWdubWVudC0xIiwiaWF0IjoxNTE2MjM5MDIyLCJleHAiOjE3MDQwNjcyMDAsInJvbGxfbm8iOiIyMHh4eHh4IiwiZW1haWwiOiJhcnVuQGlpaXRkLmFjLmluIiwiaGludCI6Imxvd2VyY2FzZS1hbHBoYW51bWVyaWMtbGVuZ3RoLTUifQ.5RcJW1ZV5gsCmV-3mufIieogVoAqr_xdyUbvLJh49dQ

We found from the payload that the secret will be alphanumeric lowercase with length of 5 characters.

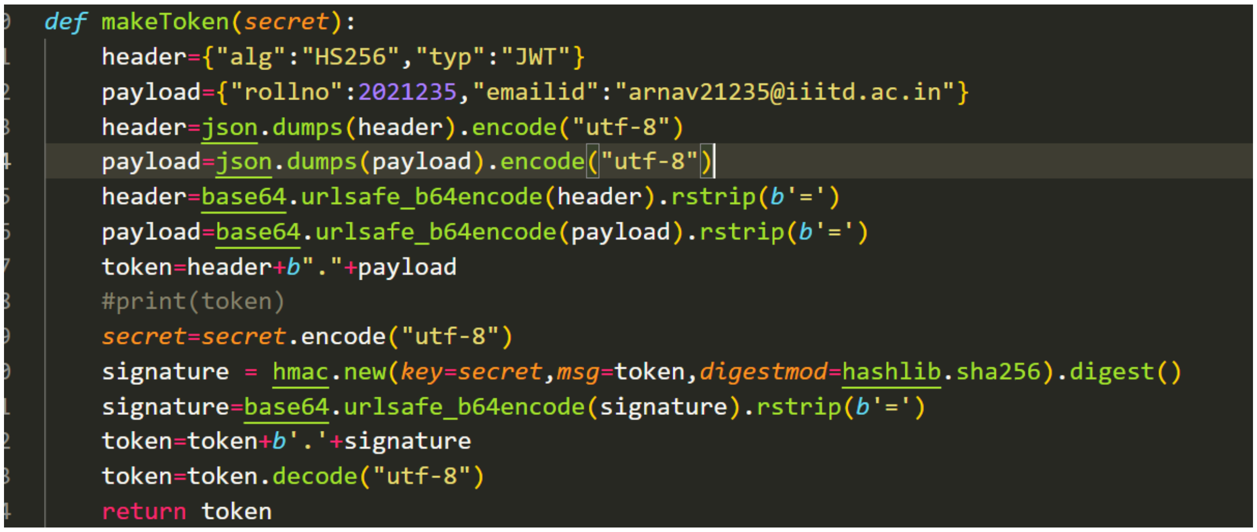
Since the length of the secret is so small we can do a brute force attack to find out the secret. Such a brute force attack took about 1 hour and 30 minutes to execute.



```
l="abcdefghijklmnopqrstuvwxyz1234567890"

    # k=0
    # permutation_list=[]
    #for perm_indices in product(range(len(l)), repeat=5):
    #     permutation = ''.join(l[i] for i in perm_indices)

    #     print(permutation)
    #     verifyJwt(secretToken,permutation)
    #     if len(validKey)>0:
    #         break
```

We use a string of length 36(26 alphabets and 10 numbers) and try out all permutations with this characters of length 5.This permutations are generated using the itertools library. verifyJWT is called for each of these secrets. The one for which the payload is returned instead of the error message is our required secret and break out of the loop.
In our case the secret was **gg476.**

We make a token containing our header and the payload as a json object and encode it into base64url.We also encode the secret and use this to create the signature using the HS256 symmetric algorithm.The signature is also base64 encoded and added to the header and payload to create the final token which is converted from bytes to plaintext and returned.



```
def makeToken(secret):
    header={"alg":"HS256","typ":"JWT"}
    payload={"rollno":2021235,"emailid":"arnav21235@iiitd.ac.in"}
    header=json.dumps(header).encode("utf-8")
    payload=json.dumps(payload).encode("utf-8")
    header=base64.urlsafe_b64encode(header).rstrip(b'=')
    payload=base64.urlsafe_b64encode(payload).rstrip(b'=')
    token=header+b"."+payload
    #print(token)
    secret=secret.encode("utf-8")
    signature = hmac.new(key=secret,msg=token,digestmod=hashlib.sha256).digest()
    signature=base64.urlsafe_b64encode(signature).rstrip(b'=')
    token=token+b'.'+signature
    token=token.decode("utf-8")
    return token
```

3)Some of the use cases of JWTs are mentioned below:
1)**Secure Communication:** the JWT could be used with an asymmetric key algorithm to sign a token with the private key so that it can be decoded with the public key. Thus we can authenticate that the token has come from the correct source and message has not been tampered with.

2)**Single Sign On:** The most popular use of the JWT is authentication mechanism called Single Sign On(SSO) across multiple applications. When a user logs into an application, they are issued with a JWT which can be used by other applications to authenticate him without requiring another login.
3)**Authorization and Authentication Systems:**Upon signing into a server generates a JWT which is sent to the client and stored by them in cookies. Before any further access to protected resources the client sends the JWT to the server which can verify the JWT against a secret to ensure that it hasn’t been tampered with and take authorisation decisions based on the JWT payload to determine if the client has required permissions to access the protected resource.
4)**Session Management:** In addition to authorization information JWTs can also store session information like the last activity of the user or a timestamp of last login. Such JWTs have a short lifespan and a new one needs to be obtained every 15 minutes or so.This reduce storage requirements of the server and allows for the server to be stateless.

These are some changes to JWT architecture to improve security:
1)**Encryption**:Headers and Payload in the JWT are inherently readable by simple base64 decoding so it is better to have some support for encryption of these fields using a public/private key to ensure confidentiality. In case not encrypted don’t store sensitive data in tokens.
2)**Token Expiration:** Tokens should be short lived to protect from token theft and compromise of tokens. Token refresh mechanisms should be implemented to generate new tokens after old ones expire.
3)**Randomness:**Secrets used to sign tokens and the tokens should be generated with strong randomness to protect from brute force attacks. The public-private key chosen should be strong.Implement good token rotation policies.