FCS REPORT

Yashasvi Chaurasia
2020159

Question 2

a) Code provided in file. 2020159_q2.py

b) The provided JWT:

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiJmY3MtYXNzaWdubWVudC0xIiwiaWF0IjoxNTE2MjM5MDIyLCJleHAiOjE2NzI1MTE0MDAsInJvbGUiOiJ1c2VyIiwiZW1haWwiOiJhcnVuQGlpaXRkLmFjLmluIiwiaGludCI6Imxvd2VyY2FzZS1hbHBoYW51bWVyaWMtbGVuZ3RoLTUifQ.LCIyPHqWAVNLT8BMXw8_69TPkvabp57ZELxpzom8FiI

Steps:

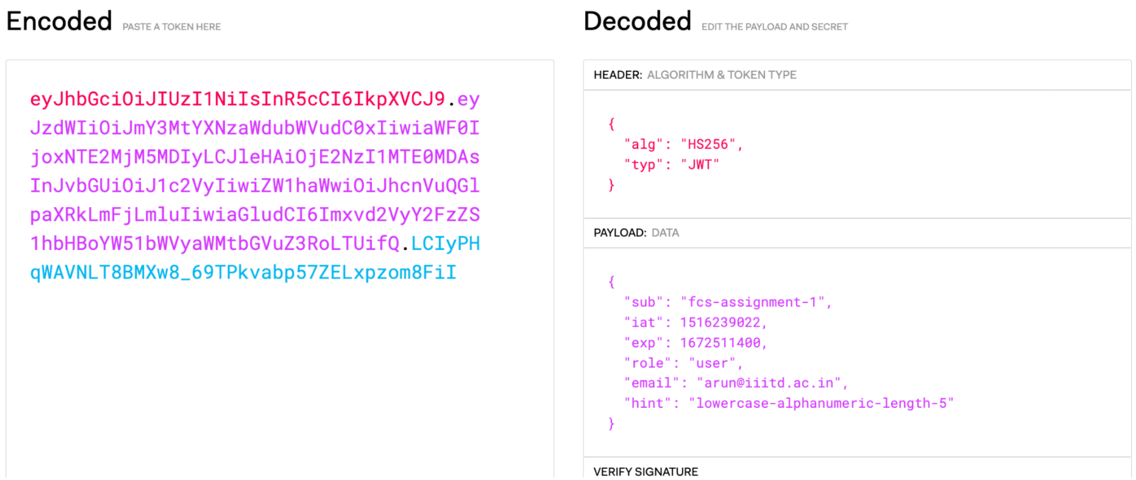
1. I plugged in the token to the jwt debugger which decoded the payload and provided the hint that the secret key used is 5 digit alphanumeric made up of lowercase letters.

2. Which meant a total of 26+10 chars so 36 characters and as we have 5 digits so the total possibility of the secret key is 36**5 ~= 60 Million which can be easily brute-forced by any modern computer in a matter of hours.

3. I then developed a function to check the valid token by brute-forced keys and return the valid key once the key is found.

4. I ran my code and it took me 10 minutes to brute force the key and find the solution to the secret key. The code is provided as “crackCode.py”

5. I then created a new jwt token with a newly modified and corrupted payload with the brute-forced key so that when the jwt token is verified the token comes out to be correct.



Encoded PASTE A TOKEN HERE

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.ey
JzdWIiOiJmY3MtYXNzaWdubWVudC0xIiwiaWF0I
joxNTE2MjM5MDIyLCJleHAiOjE2NzI1MTE0MDAs
InJvbGUiOiJ1c2VyIiwiZW1haWwiOiJhcnVuQGl
paXRkLmFjLmluIiwiaGludCI6Imxvd2VyY2FzZS
1hbHBoYW51bWVyaWMtbGVuZ3RoLTUifQ.LCIyPH
qWAVNLT8BMXw8_69TPkvabp57ZELxpzom8FiI
```

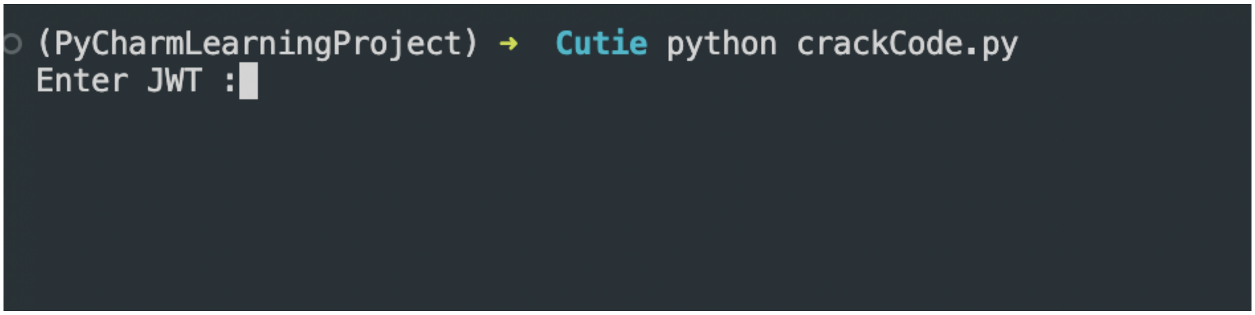
Decoded EDIT THE PAYLOAD AND SECRET

HEADER: ALGORITHM & TOKEN TYPE

```
{
  "alg": "HS256",
  "typ": "JWT"
}
```

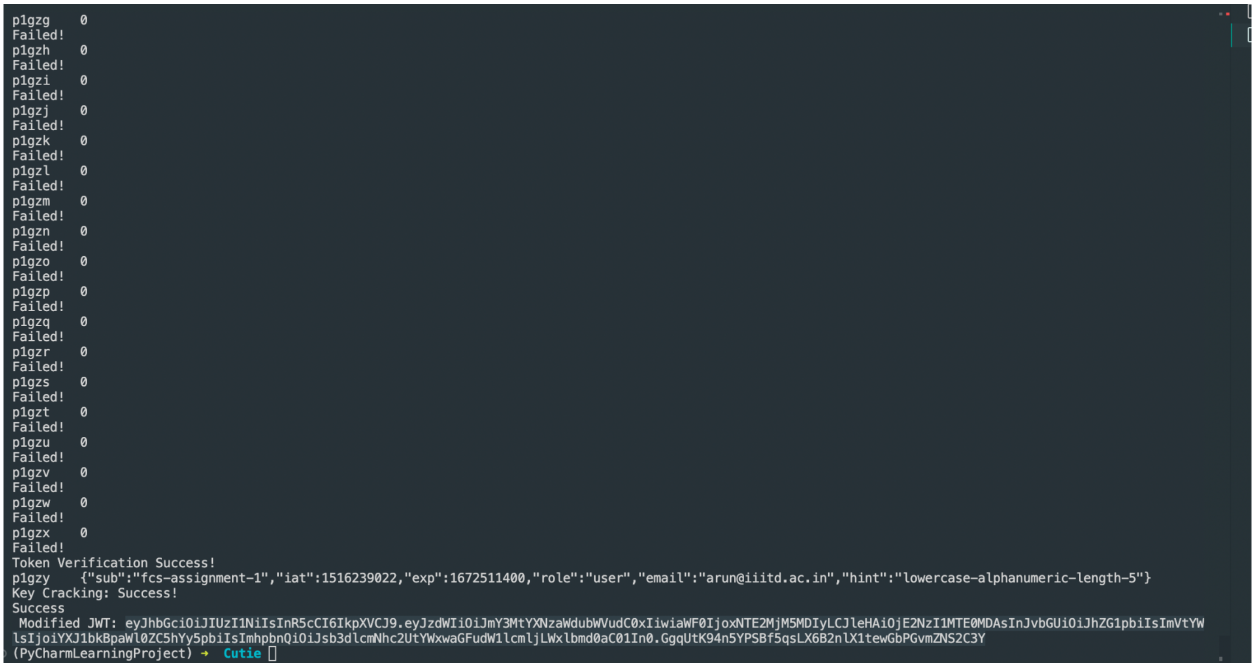
```
{
  "sub": "fcs-assignment-1",
  "iat": 1516239022,
  "exp": 1672511400,
  "role": "user",
  "email": "arun@iiitd.ac.in",
  "hint": "lowercase-alphanumeric-length-5"
}
```

Before decoding



```
○ (PyCharmLearningProject) ➜  Cutie python crackCode.py
  Enter JWT :█
```

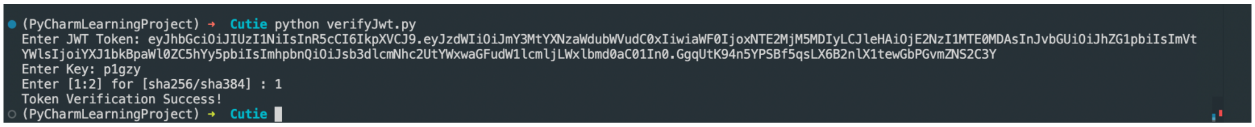
After decoding:



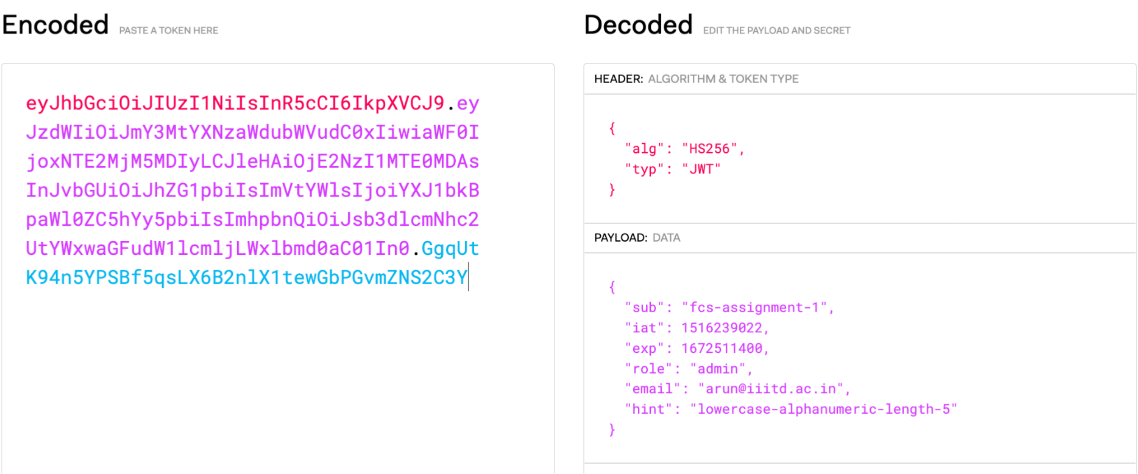
```
p1gzg    0
Failed!
p1gzh    0
Failed!
p1gzi    0
Failed!
p1gzj    0
Failed!
p1gzk    0
Failed!
p1gzl    0
Failed!
p1gzm    0
Failed!
p1gzn    0
Failed!
p1gzo    0
Failed!
p1gzp    0
Failed!
p1gzq    0
Failed!
p1gzr    0
Failed!
p1gzs    0
Failed!
p1gzt    0
Failed!
p1gzu    0
Failed!
p1gzv    0
Failed!
p1gzw    0
Failed!
p1gzx    0
Failed!
Token Verification Success!
p1gzy    {"sub":"fcs-assignment-1","iat":1516239022,"exp":1672511400,"role":"user","email":"arun@iiitd.ac.in","hint":"lowercase-alphanumeric-length-5"}
Key Cracking: Success!
Success
 Modified JWT: eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiJmY3MtYXNzaWdubWVudC0xIiwiaWF0IjoxNTE2MjM5MDIyLCJleHAiOjE2NzI1MTE0MDAsInJvbGUiOiJhZG1pbiIsImVtYW
lsIjoiYXJ1bkBpaWl0ZC5hYy5pbiIsImhpbnQiOiJsb3dlcmNhc2UtYWxwaGFudW1lcmljLWxlbmd0aC01In0.GgqUtK94n5YPSBf5qsLX6B2nlX1tewGbPGvmZNS2C3Y
○ (PyCharmLearningProject) ➜  Cutie ▯
```

The function return the new corrupted but valid token and the secret key .

The new corrupted token is then verified:



```
● (PyCharmLearningProject) ➜  Cutie python verifyJwt.py
  Enter JWT Token: eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiJmY3MtYXNzaWdubWVudC0xIiwiaWF0IjoxNTE2MjM5MDIyLCJleHAiOjE2NzI1MTE0MDAsInJvbGUiOiJhZG1pbiIsImVt
  YWlsIjoiYXJ1bkBpaWl0ZC5hYy5pbiIsImhpbnQiOiJsb3dlcmNhc2UtYWxwaGFudW1lcmljLWxlbmd0aC01In0.GgqUtK94n5YPSBf5qsLX6B2nlX1tewGbPGvmZNS2C3Y
  Enter Key: p1gzy
  Enter [1:2] for [sha256/sha384] : 1
  Token Verification Success!
○ (PyCharmLearningProject) ➜  Cutie █
```



```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.ey
JzdWIiOiJmY3MtYXNzaWdubWVudC0xIiwiaWF0I
joxNTE2MjM5MDIyLCJleHAiOjE2NzI1MTE0MDAs
InJvbGUiOiJhZG1pbiIsImVtYWlsIjoiYXJ1bkB
paWl0ZC5hYy5pbiIsImhpbnQiOiJsb3dlcmNhc2
UtYWxwaGFudW1lcmljLWxlbmd0aC01In0.GgqUt
K94n5YPSBf5qsLX6B2nlX1tewGbPGvmZNS2C3Y
```

```
{
  "alg": "HS256",
  "typ": "JWT"
}
```

```
{
  "sub": "fcs-assignment-1",
  "iat": 1516239022,
  "exp": 1672511400,
  "role": "admin",
  "email": "arun@iiitd.ac.in",
  "hint": "lowercase-alphanumeric-length-5"
}
```

c)
The problem with the current jwt token is that if a secret key is leaked, then any malicious user can corrupt the file by modifying the file and again appending the correct signature with it. Hence reusing a secret key poses a confidentiality and integrity threat.
Instead of using the same key again, we can combine hash chaining with jwt to generate new keys every time the server and the client communicate.

Explanation of the proposal:

The hash seed is kept with the server, which uses it for all communications throughout the day.

The seed is used to generate keys which will be used secret for JWT tokens.

Initially, the server uses a key from the hash-chaining dictionary in reverse order.

Once the token completes one round trip, the server verifies the token from the supplied key, and then it supplies a new token to the client which uses the secret key which is generated via hash chaining.

This mode of generating keys is secure as the seed is never transmitted over any network and it is always securely kept with the server. Also even if an attacker is able to brute force the secret key then they cannot reuse the secret key as it is changed after every round trip to the server. And there is no way of guessing the next key.

It is easy for the server to maintain a list of keys every time as we have already precomputed the secret keys using hash chaining and we use them in reverse order to avoid anyone from predicting the next key.