


# FCS ASSIGNMENT-1

## REPORT FOR QUESTION 2

### Answer for part b:

In order to find the secret key for the given JWT, the following steps were performed:

- I first entered the JWT on the website [jwt.io](https://jwt.io). From there, the following was found out:

HEADER: ALGORITHM & TOKEN TYPE	
	<pre>{   "alg": "HS256",   "typ": "JWT" }</pre>
PAYLOAD: DATA	
	<pre>{   "sub": "fcs-assignment-1",   "iat": 1516239022,   "exp": 1672511400,   "role": "user",   "email": "arun@iiitd.ac.in",   "hint": "lowercase-alphanumeric-length-5" }</pre>
VERIFY SIGNATURE	
<pre>HMACSHA256(   base64UrlEncode(header) + "." +   base64UrlEncode(payload),   your-256-bit-secret ) <input type="checkbox"/> secret base64 encoded</pre>	

- From here I saw the hint in the payload and tried to figure out that it may be possible to brute force the key as it consists of only lowercase letters and digits. For this I wrote a python code using the 'PyJWT' library. The code is as follows:

```

token =
"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiJmY3MtYXNzaWdu
bWVudC0xIiwiaWF0IjoxNTE2MjM5MDIyLCJleHAiOjE2NzI1MTE0MDAsInJvG
UiOiJ1c2VyIiwiaWlhaWwiOiJhcnVuQGlpYXRkLmFjLmluIiwiaGludCI6Imxv
d2VyY2FzZS1hbHB0YW51bWVyaWMtbGVuZ3RoLTUifQ.LCIyPHqWAVNLT8BMXw8
_69TPkvabp57ZELxpzom8FiI"

def verify(k):
    try:
        x=jwt.decode(token, key=k, algorithms=['HS256', ])
        # print(x)
        return x
    except:
        return -1
words=[]

chars = "qwertyuiopasdfghjklzxcvbnm0123456789"
n=36

def printAllKLengthRec(prefix, k):

    # Base case: k is 0,
    # print prefix
    if (k == 0) :
        words.append(prefix)
        return

    # One by one add all characters
    # from set and recursively
    # call for k equals to k-1
    for i in range(n):
        # Next character of input added
        newPrefix = prefix + chars[i]

        # k is decreased, because
        # we have added a new character
        printAllKLengthRec(newPrefix, k - 1)
printAllKLengthRec("", 5)
i=0
m=len(words)

```

```
y=-1
while y== -1 and i<m:
    y=verify(words[i])
    i+=1

print(words[i-1])
print()
print(y)
```

Logic: I first found out all the possible 5 letter strings possible using the recursive function `printAllKLengthRec()`. After storing them in a list, I went over the list and tried whether any of the strings can be used to decode the token (as in `verify()` function using `PyJWT` library).

The output of the code was:

p1gzy

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

- Hence the secret key found out is: 'p1gzy'.
- While reading about cracking JWTs [here](#), I also found this [repo](#) which decodes the JWT.

```
[vishesh@fcs01:~/FCS$ ls  
c-jwt-cracker  
[vishesh@fcs01:~/FCS$ cd c-jwt-cracker/  
[vishesh@fcs01:~/FCS/c-jwt-crackers$ ls  
Dockerfile Makefile base64.c base64.o jwtcrack main.o  
LICENSE README.md base64.h entrypoint.sh main.c  
[vishesh@fcs01:~/FCS/c-jwt-crackers$ time ./jwtcrack eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiJmY3MtYXNzaWdubWVudC0xiiwiaWF0IjoxNTE2MjM5MDIyLClleHAiOjE2ZnI1MTE0MDAsInR5cGUlOiJ1c2VyTiwiZW1haWwibGludC16Imxvd2VyZFZ5ZS1hbHoYS1biWVyaWMtbGVuZ3RoLTUIfQ.LCiYPHqMAVNLT8BMXw8_69TPkvabp57ZELxpzm8FiI  
Secret is "plgzy"  
  
real    23m0.584s  
user    45m54.879s  
sys     0m0.964s  
vishesh@fcs01:~/FCS/c-jwt-crackers$
```

- After entering this secret key on [jwt.io](https://jwt.io), even that showed signature verified:



Answer for part c:

A more secure authentication measure should be adopted in order to prevent the widespread damage in the case of leakage of the key. To accomplish this, instead of using 1 secret key, multiple secret keys could be used in rotation. So even if one of the keys gets leaked all of the information cannot be tampered with.

Even these keys should be stored at decentralized locations to prevent their leak. Its also better to store the keys in encrypted form.

Another architectural change could be that one should use an asymmetric cryptography algorithm for setting up communication channel. This channel should then be used for data exchange using the shared secret key. This way even if an attacker gets your shared secret key, there would be communication over a secure channel and the attacker won't be able to access your data exchange.

References:

<https://auth0.com/blog/rs256-vs-hs256-whats-the-difference/>