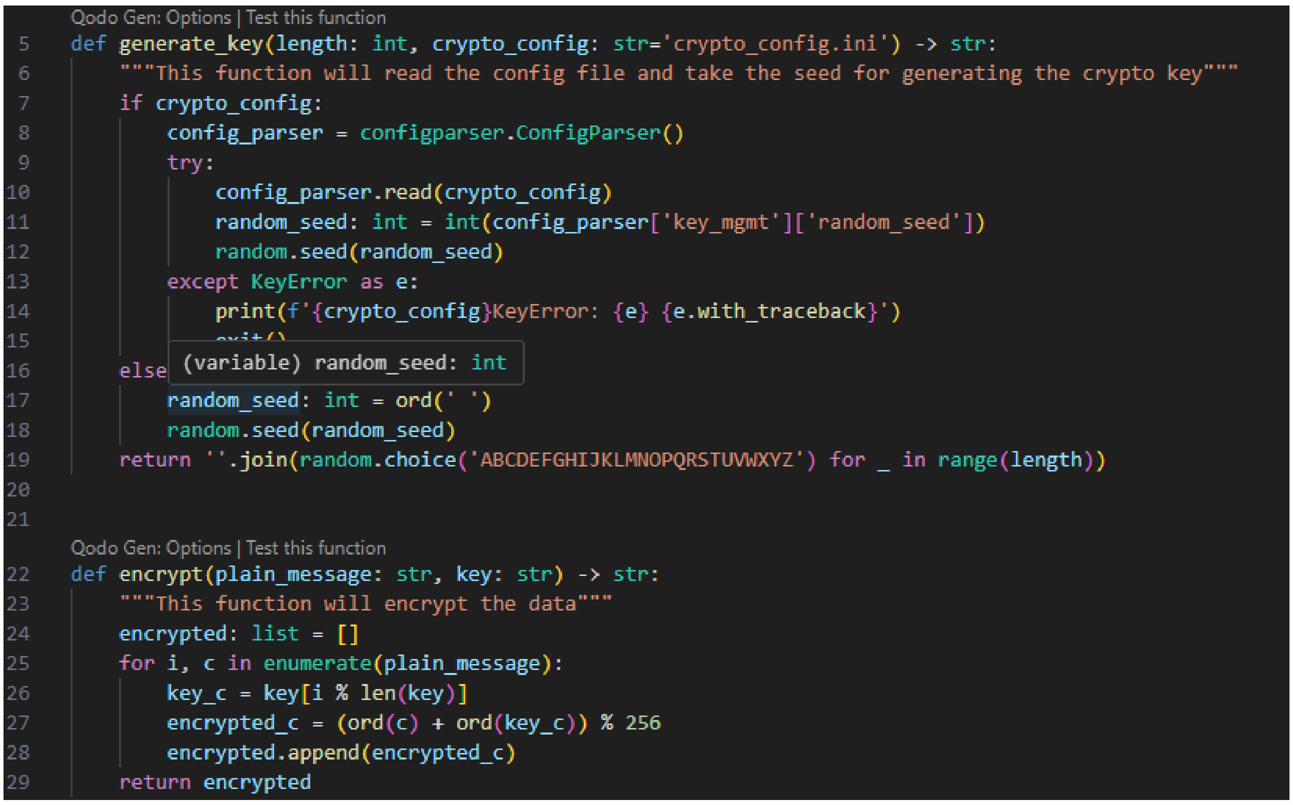
Crypt(o)

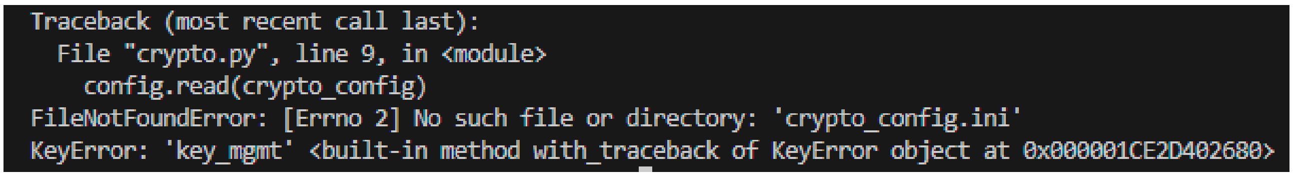
DarkTemplar
Kleopatras_Klantarslen

Right at the start of this challenge we get to download a file called crypto.py. Seems like it is a python-file. But never take anything for granted in a CTF. Anyway, we download it and try to open it with Visual Studio Code. And it indeed is python.



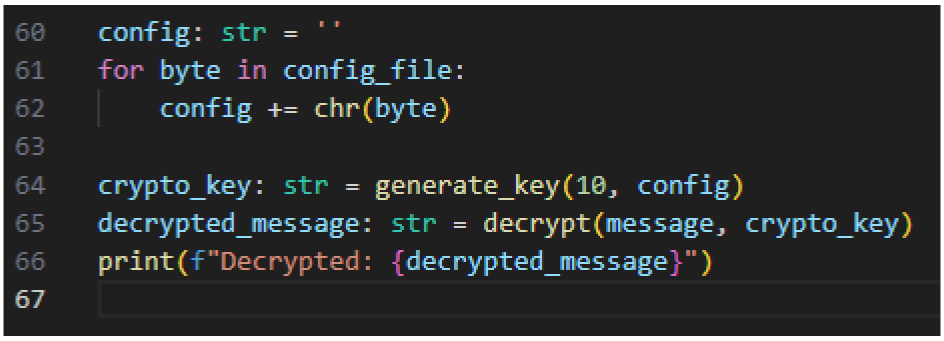
```
Qodo Gen: Options | Test this function
def generate_key(length: int, crypto_config: str='crypto_config.ini') -> str:
    """This function will read the config file and take the seed for generating the crypto key"""
    if crypto_config:
        config_parser = configparser.ConfigParser()
        try:
            config_parser.read(crypto_config)
            random_seed: int = int(config_parser['key_mgmt']['random_seed'])
            random.seed(random_seed)
        except KeyError as e:
            print(f'{crypto_config}KeyError: {e} {e.with_traceback}')
            exit()
    else (variable) random_seed: int
        random_seed: int = ord(' ')
        random.seed(random_seed)
    return ''.join(random.choice('ABCDEFGHIJKLMNOPQRSTUVWXYZ') for _ in range(length))


Qodo Gen: Options | Test this function
def encrypt(plain_message: str, key: str) -> str:
    """This function will encrypt the data"""
    encrypted: list = []
    for i, c in enumerate(plain_message):
        key_c = key[i % len(key)]
        encrypted_c = (ord(c) + ord(key_c)) % 256
        encrypted.append(encrypted_c)
    return encrypted
```

Let’s try to figure out what it does. A simple run of the script gives us this output



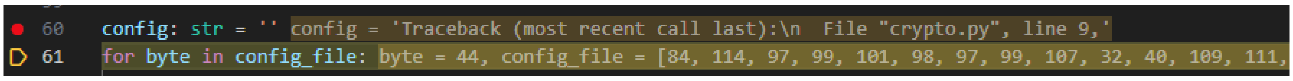
```
Traceback (most recent call last):
  File "crypto.py", line 9, in <module>
    config.read(crypto_config)
FileNotFoundError: [Errno 2] No such file or directory: 'crypto_config.ini'
KeyError: 'key_mgmt' <built-in method with_traceback of KeyError object at 0x000001CE2D402680>
```

So now we know what the original output is. Let’s scour the code and see if we find anything interesting.



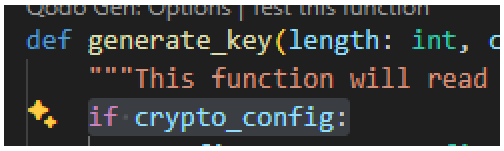
```python
config: str = ''
for byte in config_file:
    config += chr(byte)

crypto_key: str = generate_key(10, config)
decrypted_message: str = decrypt(message, crypto_key)
print(f"Decrypted: {decrypted_message}")

```

Down at the bottom we see the function calls. But before that we see a for loop that iterates through a list of ints and turns them into characters and stores them in the variable config. This config variable then gets sent in to the generate_key function. Let's debug it and see what the list gets converted to.



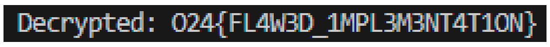
```
config: str = '' config = 'Traceback (most recent call last):\n  File "crypto.py", line 9,'
for byte in config_file: byte = 44, config_file = [84, 114, 97, 99, 101, 98, 97, 99, 107, 32, 40, 109, 111,
```

Hold on a sec! After some iterations we start seeing what get's converted from the list. And doesn't it look familiar? That seems to be the same error message we got when we just ran the script.



```python
Qodo Gen: Options | Test this function
def generate_key(length: int, c
    """This function will read
    if crypto_config:
```

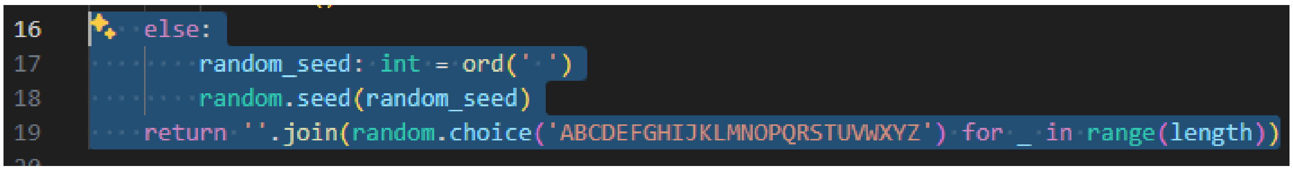
If we inspect this line of code a bit. It says that if there is a value in crypto_config it will execute that part of the code. But if there is no value. Or a boolean False. It will not be executed. What if we just comment out the for loop that gives the variable config it's value. Then this statement would evaluate as False and not be executed. Worth a shot to see what happens.



```
Decrypted: O24{FL4W3D_1MPL3M3NT4T1ON}
```

Bingo! We get our flag.

If you want an explanation of what is happening from my understanding, just keep reading.



```python
    else:
        random_seed: int = ord(' ')
        random.seed(random_seed)
    return ''.join(random.choice('ABCDEFGHIJKLMNOPQRSTUVWXYZ') for _ in range(length))
```

If we look into this part of the code, which is the one that gets executed when if evaluates as False. We see that it creates a variable named random_seed and sets it to the value of a space in char code. This means that the value of random_seed gets set to 32. And when random.seed(random_seed) gets run. It sets the seed that random should use for its randomisation.

In the return statement, we see that random.choice gets used to pick out a character from the string which includes the english alphabet in capital letters. This gets looped as many times as the value of length. Which is 10 since that is the first parameter of the function, and the function call is generate_key(10, config). As the seed is set to a fixed value. The randomisation won't actually be random at all. It will always choose the same ten letters. And therefore later on in the code when it decrypts the message, it will decrypt correctly.