Factor's:

1. Factorization of n is possible due to smaller p and q values - used

https://www.alpertron.com.ar/ECM.HTM

```
def extended_gcd(a, b):
    x0, x1, y0, y1 = 1, 0, 0, 1
    while b != 0:
        q, a, b = a // b, b, a % b
        x0, x1 = x1, x0 - q * x1
        y0, y1 = y1, y0 - q * y1
    return a, x0, y0


def mod_inverse(a, m):
    g, x, y = extended_gcd(a, m)
    if g != 1:
        raise Exception('Modular inverse does not exist')
    else:
        return x % m


# Given values
n = ...   # modulus (not shown fully in your image)
e = 65537 # public exponent
c = ...   # ciphertext

# Factorized primes
p = 15485863
q =
263840088670917452946333545478352127416911466730977444594879161708660689824619163128492286549101212418432724327154652577505575308088758980984808941611740021708982184108224975505577336225957797888187213725467913138087565613760703636137433576401356908527651990944725582431699173155977628169591056426990285120277950325598
```

```
00770588152330655774546219611361677479090675113787095736605672768
62339597448484330993224046774434020874200594041033926202578941508
96959622939815996558152312664311513 7")

# Verify n = p * q
if n != p * q:
    print("Error: n != p * q")
    exit(1)

# Compute Euler's totient function
phi = (p - 1) * (q - 1)

# Compute private key d
d = mod_inverse(e, phi)

# Decrypt the message
m = pow(c, d, n)

# Convert decrypted number to bytes
message = m.to_bytes((m.bit_length() + 7) // 8, 'big')
print(message.decode())


```

Final FLAG:
Flag{kiet_sh0uld'v3_t4k3n_b1gg3r_pr1m3s_xd}