Factor's:

```
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 = ...
e = 65537
c = ...
# Factorized primes
p = 15485863
q =
26384008867091745294633354547835212741691416673097444594871961708606898246191631284922865941012124184327243247514562575750057530808887589809848089461174100421708982184082294675500577336225957797988818721372546749131380876566137607036301473435764031659085276159909447255824316991731559776281695919056426990285120277950325598700770588152330565774546219611360167747900967511378709576366056727866239359744484343099322440674434020874200594041033926202578941508969596229398159965581521326643115137
# Verify that 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 the private key d
d = mod_inverse(e, phi)
# Decrypt the message
m = pow(c, d, n)
# Convert the message to bytes (assuming it's a string of bytes)
```

```
message = m.to_bytes((m.bit_length() + 7) // 8, 'big')
print(message.decode())
```

Final FLAG:
flag{kiet_sh0uld'v3_t4k3n_b1gg3r_pr1m3s_xd}