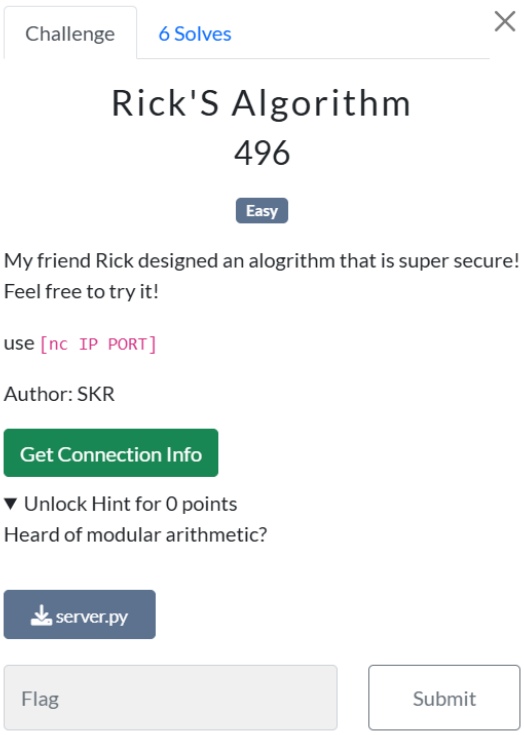
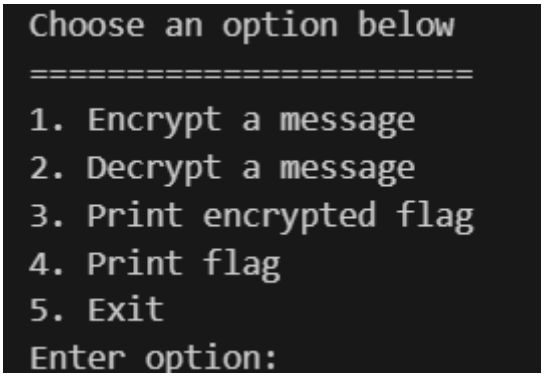
Rick’S Algorithm Writeup



We will be greeted with this program which able to do this process:



After some careful reading of the code given (server.py), we can see that there are two condition (c % pow(flag,e,n) == 0) which prevents user to directly insert the encrypted flag to the decryption oracle and if the flag == decrypted text (flag % pow(c,d,n) == 0), it will break the process which doesn’t allow us to decrypt it

```
elif option == "2":
    c = int(input("Enter ciphertext to decrypt: "))
    if c % pow(flag,e,n) == 0 or flag % pow(c,d,n) == 0:
        print("HACKER ALERT!!")
        break
    print(f"Decrypted message: {pow(c,d,n)}")
```

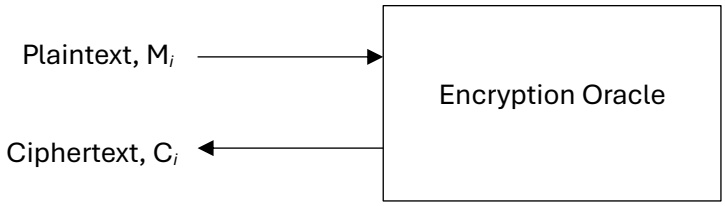
We also can see the encryption process:

```
option = input("Enter option: ")
if option == "1":
    m = bytes_to_long(input("Enter message to encrypt: ").encode())
    print(f"Encrypted message: {pow(m,e,n)}")
```

Based on this GitHub, https://github.com/ashutosh1206/Crypton/blob/master/RSA-encryption/Attack-Retrieve-Modulus/README.md

We can try CPA Attack to find n value

In Chosen Plaintext Attack, we will try to choose and send some plaintext to the encryption oracle to get back the ciphertext



If e is known (e=0x557), we can compute n which comes from the formula:

$n = GCD(C_i - M_i^e, C_{i+1} - M_{i+1}^e)$ // if there are only two pairs of plaintexts and ciphertexts

In this writeup, I choose '!' as M_1 and 'B' as M_2

```
# CPA attack (Chosen Plaintext Attack)
# plaintext -> encryption oracle -> ciphertext
# Choose 2 plaintext to get 2 ciphertext
m1 = 33 # ! in ascii
m2 = 66 # B in ascii
```

Get ciphertext C_1 and C_2:

```
c1 =
220301332429641667977307083990123384050303574588369567265219765750939563429021194709312901605629188458133727004254331288514755983183066585199903929508788450778265805258515259023829790618922736647306261299464789819529757345626208212084610489270368866348374509648007695033889816513170504679798670432769845868299739581860290034362472242126581853332502142651577163808493371115625226011183928076308438873306416948363037489586466204369598731610467680452402918003821741564580350076154873240855112993268826865285416562538355786449087564863892603550461180695735159426188912568988494928768573753953875853116608935851267941564780
c2 =
121255108484210106346854746054976218382463209894206162523954369432754826135185378102447621558483698075637576982092272745650572298361756249300704210393983126777494573951036911212354645034555775271177415348191345766968801374164109591641357505506031376749728499854149792228206116479629878655466623071068829428342122849774204664586285068270825445481613308743680029947039276915415445906361343437645929011057935399427976602033191701967821258344772348322361185863336947201877984163502298340425973564613978331231866638689129329524787465615468201221446348733176198530068502863053288874295173338894335474569745683135504535063660
```

Compute n:

```
# Compute n = gcd(c1-m1^e,c2-m2^e)
first = c1 - pow(m1, e)
second = c2 - pow(m2, e)
n = math.gcd(first, second)
print("N: ",n)
```

After getting the n, we can try to interfere with the two conditions for decryption oracle. Based on the first condition (c % pow(flag,e,n) == 0), we can break this condition by adding the n value with the encrypted flag value.

$$new\ encrypted\ flag = encrypted\ flag + n$$

$$new\ encrypted\ flag\ \%\ pow(flag, e, n)$$

$$new\ encrypted\ flag\ \%\ encrypted\ flag\ != 0$$

But, by using this method, it cannot break the second condition (flag % pow(c,d,n) == 0).

$new\ encrypted\ flag = encrypted\ flag + n$

$flag\ \%\ pow(new\ encrypted\ flag, d, n)$

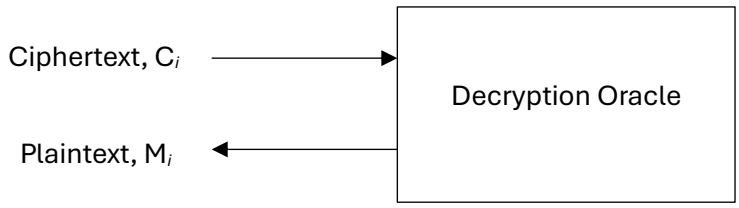
$decrypted\ flag = (new\ encrypted\ flag)^d\ mod\ n$

//n value inside the new encrypted flag will become 0 after mod n.

$flag\ \%\ decrypted\ flag\ == 0$

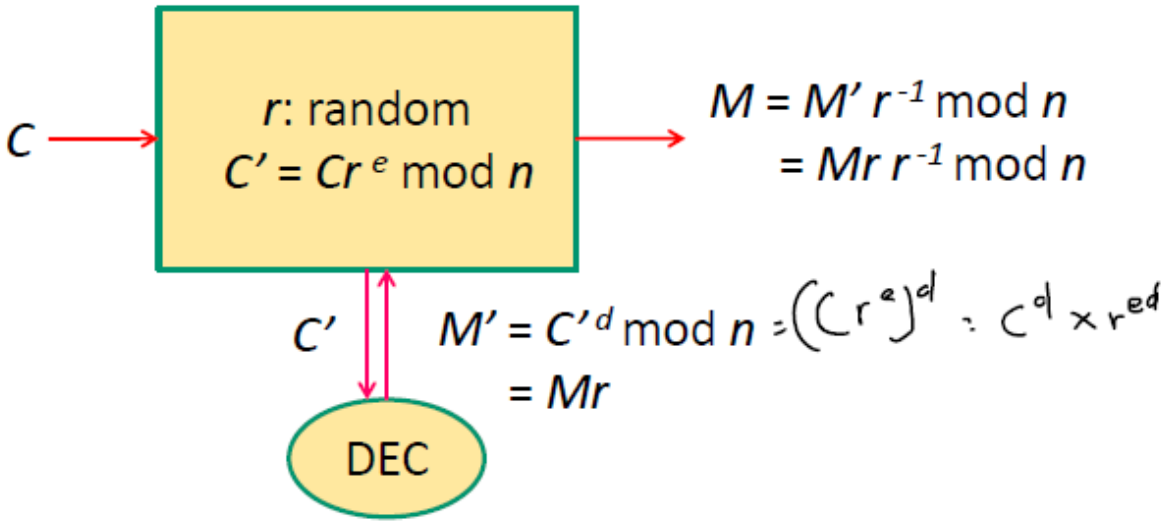
So, we need to come up with a solution to break both the conditions.

I come up with CCA Attack which is a Chosen Ciphertext Attack:



In CCA, we will choose ciphertext to be sent to the decryption oracle to get the plaintext.

By taking the advantage of the multiplicative property of RSA, we can perform as below:



We can find a random value ‘r’ which is co-prime (relative prime) to n value.

```
# Function to generate random `r` value from n
def generate_random_r(n):
    while True:
        r = randint(2, n - 1)  # Random integer in range [2, n-1]
        if gcd(r, n) == 1:  # Ensure `r` is coprime with `n`
            return r

# Generate random `r` value
r = generate_random_r(n)
print("\nRandom r value:", r)
```

Then. we can find C’(new encrypted flag) by having the C(encrypted flag), r, e, and n values by:

$$C' = Cr^e \ mod \ n$$

```
# C' = C * r^e mod n
new_encryptedflag= (encrypted_flag*pow(r, e)) % n
```

```
New Encypted Flag (C'):
9221670609472587060886419739786337400227440212092971258863371447412282290021220805926169
2855705545400990320942248382763210505198326996461088569629504867553119679382725098624865
1718805046240240925697511292279569129995239343222008736158400868675064439380738870413827
3521615308616131574314083344617652722176301203516988365309671674802236036485165322908283
3720506054957955500702242557738773722083428822066020179453516749275352595147827176571197
9475147717949090731213514624359934253538023750074785553819459159983398624845487359572481
489418926420954267350047575364855825356484496507442831806786932685015300653551445356750
```

After we have done getting C’, send the C’ to the decryption oracle to get M’.

```
# Receive M' from the decryption oracle
# M' = M * r
m_prime =
1593758586364472655727943738413723224321086583772538762859506868188461972
6105021399311715128422430650551422880337525017496385627885613730540850481
4155487571201838806497235781056326638778323838867877963408808227758760772
3692803484421686946655715643204729106941171337930247488379984127624424423
3859610205735135902395295308897956560304863397181446639418735707766802827
4357420323621709555436711504245303974355775254264071271013416105663079182
3287271111187824805295267849505901082774501807983750075105006180342085527
9204266100951792607180252602896100008517843721890788225101029765099138230
098561232375796899337140243476730
```

M' is equivalent to:

$M' = C'^{d} \ mod \ n$

$M' = (Cr^{e})^{d} \ mod \ n$

$M' = (C^{d} \ x \ r^{ed}) \ mod \ n$

//in RSA, e*d will always equal to 1 and M = C^{d} mod n

$M' = (M \ x \ r) \ mod \ n$

$M' = Mr$

$M = M'r^{-1} \ mod \ n$

$M = Mr \ x \ r^{-1} \ mod \ n$

So, to get the flag (which is M), we need to multiply M' with r^{-1}. We also can use the modular inverse function to find M:

```
# M = M' * r^-1 mod n
encoded_flag = (m_prime * mod_inverse(r,n)) % n
```

```
Encoded Flag:
15201892040119364990483834841798236623598320919650209676780920528532794290727504714893714557
```

But the output seems to be encoded. Decode the encoded flag from long to bytes:

```
print("\nDecoded Flag:",long_to_bytes(encoded_flag).decode())
```

```
Decoded Flag: wgmy{ce7a475ff0e122e6ac34c3765449f71d}
```

Flag: `wgmy{ce7a475ff0e122e6ac34c3765449f71d}`