AdveRSArial Crypto CTF Writeup

This document is a walkthrough on one way to solve the **AdveRSArial Crypto CTF** on **Hackropole**. The objective is to explain how I was able to solve this CTF to my future self.

General Information

- *Difficulty:* Easy/Medium
- *Category:* Cryptography
- *Link:* [AdveRSArial Crypto - Hackropole]

Solution

We're given the following python script and text file:

adversarial-crypto-infant_solution.py × adversarial-crypto-infant.py ×

```python
from Crypto.Util.number import getStrongPrime, bytes_to_long, long_to_bytes

n = getStrongPrime(2048)
e = 2 ** 16 + 1

flag = bytes_to_long(open("flag.txt", "rb").read())
c = pow(flag, e, n)

print(f"{n = }")
print(f"{e = }")
print(f"{c = }")

```

```
┌──(alexandre㉿vbox)-[~/Documents/CTF Files]
└─$ strings output.txt
n = 229147643496975569635416926657210764254900639915749362435714281562613020603286855915565140367517
7777606577116733024401070808214740140200291437790495008048679995700511136036502809288436737333845422
3568447811216200859660057226322801828334633020895296785582519610777820724907394060126570265818769159
9917521447834693385576914071024327866446945901181765820009651243605002579463040287670882967249070625
6116347865499599420506581247960513608881354343589584027606668324370602009151985727521942224600613739
0619897086478975872204136389082598585864385077220265194919486850918633328368814287347732293510186569
121425821644289329813
e = 65537
c = 111899171606987386479114334936932851015381314550356115500779507091074293312983295023273585887742
6116167442235173994112088228995440047759050227262969385324211650700043376191436881465618087478359481
226049854239050022151988309947855086317214758892234157144350244943514309057651422827483316274013491
9379193979570175466713253570277658176925715839984873520907898559801311844516110878223990886697056837
6537051005278174238373627829529601226779442926372050972479442655201074167834283831906008407482671306
5120930332229122961216786019982413982114571551833129932338204333681414465713448112309599140515483842
800125894387412148599
```

In order to do this challenge, we must understand how RSA encryption works.
The following video is recommended:

That being said, let's look at the program.
It gets a big prime number n, sets e to 65537, a commonly-used value in RSA encryption, using the **"pow(flag, e, n)"** function, which is equivalent to **(flag^e) % n**, it stores its value into c

Now, we know that given the public key e, and n it's able to encrypt any message using the following algorithm: Encrypted_Message = (Message^e) % n

To proceed, we'll first need what's called Euler's Totient, which is equal to **(p-1)(q-1)**, with p, q being prime factors of n. However, since n is prime, it's totient T is simply equal to **n-1**.

To decrypt this message, we first need the private key d,
which must satisfy the following equation: d*e % T = 1
To find this, we can use the the **inverse** function in python

```
d = inverse(e, n-1)
```

Now that d has been determined, to decrypt the message, we use the following algorithm:
Message = (Encrypted_Message^d) % n

Considering that in the program, the flag is the encrypted message, we'll need to decrypt it by first determining d, and then applying the formula above.

As a result, we've written the subsequent script to solve the problem:

adversarial-crypto-infant_solution.py × adversarial-crypto-infant.py ×

```python
from Crypto.Util.number import getStrongPrime, bytes_to_long, long_to_bytes, inverse

with open('output.txt', 'r') as file:
    lines = file.readlines()

n = int(lines[0].split('=')[1].strip())
e = int(lines[1].split('=')[1].strip())
c = int(lines[2].split('=')[1].strip())

d = inverse(e, n-1)

flag = pow(c, d, n)

print(long_to_bytes(flag).decode())
```

```
FCSC{d0bf88291bcd488f28a809c9ae79d53da9caefc85b3790f57615e61c70a45f3c}
[Finished in 48ms]
```

We get the following flag:

FCSC{d0bf88291bcd488f28a809c9ae79d53da9caefc85b3790f57615e61c70a45f3c}