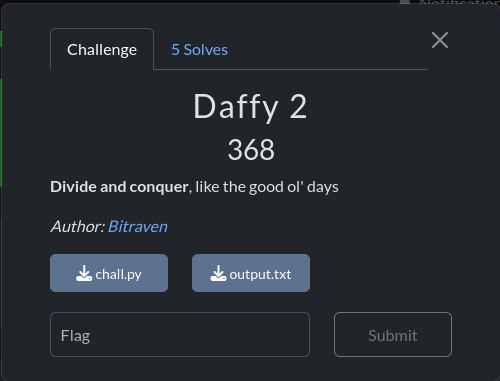
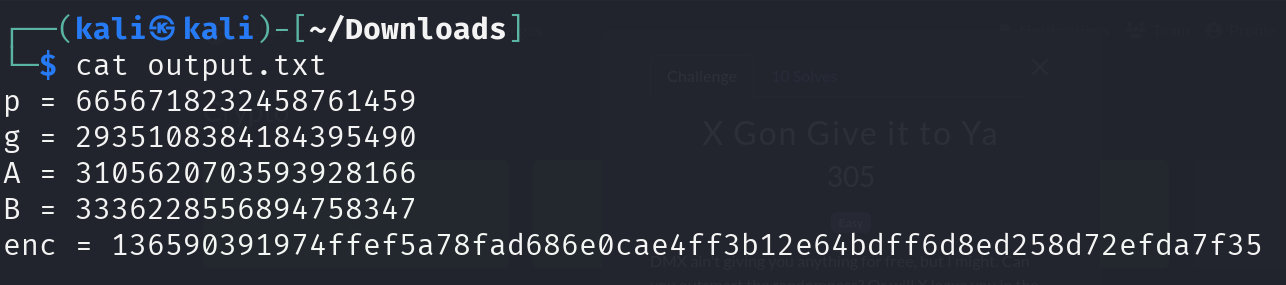
**Daffy 2**



**Understanding the Challenge**

We are given a file with some numbers and an encrypted message (enc). These numbers represent values used in a cryptographic system called **Diffie-Hellman Key Exchange (DHKE)**. Our goal is to **derive the shared secret key** and use it to decrypt the flag.



The provided Python script generates these values and encrypts the flag using **AES encryption** in **ECB mode**.

**What is Diffie-Hellman?**

Diffie-Hellman is a way for two parties to securely agree on a shared secret over an insecure channel. Here’s how it works in our challenge:

1. **Public Parameters:**
   * p: A large prime number
   * g: A generator (used to create the keys)
2. **Private Keys (Secret numbers chosen by each party):**
   * Alice picks a **private key** a
   * Bob picks a **private key** b
3. **Public Keys (Sent over the network):**
   * Alice computes A=gamod  pA = g^a \mod pA=gamodp and sends it
   * Bob computes B=gbmod  pB = g^b \mod pB=gbmodp and sends it
4. **Shared Secret Calculation:**
   * Alice computes S=Bamod  pS = B^a \mod pS=Bamodp
   * Bob computes S=Abmod  pS = A^b \mod pS=Abmodp
   * Both parties get the same **shared secret SSS**

**How We Solve It**

We are given:

* p, g, a, and b (from the challenge file)
* The encrypted flag (enc)

To decrypt the flag, we need to:

1. Compute the shared secret:

S=Bamod  pS = B^a \mod pS=Bamodp

(Since we don’t have a, we compute S=Abmod  pS = A^b \mod pS=Abmodp instead.)

1. Convert S into a **16-byte key** (since AES requires a key).
2. Decrypt the flag using **AES in ECB mode** with the derived key.

The solution this **python script** below copy, paste and then run you’ll find your flag:

import math

from Crypto.Util.number import long\_to\_bytes

from Crypto.Cipher import AES

from Crypto.Util.Padding import pad, unpad

p = 6656718232458761459

g = 2935108384184395490

A = 3105620703593928166

B = 3336228556894758347

enc = bytes.fromhex("136590391974ffef5a78fad686e0cae4ff3b12e64bdff6d8ed258d72efda7f35")

# Step 1: Recover the subgroup order (i.e. small\_prime)

order = None

for k in range(1, 70000):

    if pow(g, k, p) == 1:

        order = k

        print("small\_prime =", order)

        break

# Step 2: Baby‐step Giant‐step discrete log in a subgroup of size “order”

def discrete\_log(g, h, p, order):

    m = int(math.ceil(math.sqrt(order)))

    table = {}

    cur = 1

    for j in range(m):

        table[cur] = j

        cur = (cur \* g) % p

    # g^(-m) mod p

    inv = pow(g, p-1-m, p)

    cur = h

    for i in range(m):

        if cur in table:

            return i \* m + table[cur]

        cur = (cur \* inv) % p

    return None

a\_mod = discrete\_log(g, A, p, order)

b\_mod = discrete\_log(g, B, p, order)

print("a\_mod =", a\_mod, "b\_mod =", b\_mod)

# Step 3: Compute the shared secret S

S = pow(g, (a\_mod \* b\_mod) % order, p)

print("S =", S)

# Step 4: Derive AES key and decrypt

key = long\_to\_bytes(S)[:16]

aes\_key = pad(key, 16)

cipher = AES.new(aes\_key, AES.MODE\_ECB)

pt = unpad(cipher.decrypt(enc), 16)

print("Decrypted flag:", pt.decode())

kabooooom we find our flag ☺

