**Task 2**

**A brief description about getting the private key**

**1)** Run *get\_pri\_key.py* with GTID for Public Key (N, e)

GTID: ashen38 | N: 0xcd6340f34383bdf | e: 0x10001

**2)** Factor N into two relative primes, or P and Q

This is done by finding the largest prime of N that is smaller than its own square root. This works because RSA is designed with P and Q to be relatively proximate, and therefore maximizing factoring/brute-forcing time. I have accomplished this by using the code from (it runs for about 3 minutes) <http://stackoverflow.com/questions/15347174/python-finding-prime-factors>

\*I defined the initial result as P but we can name it either P or Q. Since N = P \* Q, Q = N / P.

**3)** d ≡ e−1 mod φ(N). As we have no readily available inverse modulo function in the Python Standard Library, we have to implement our own Extended Euclidean Algorithm. The implementation consists of finding the GCD and inversing it. This is accomplished using the code from <http://crypto.stackexchange.com/questions/19444/rsa-given-q-p-and-e>

\*Since e = 0x10001, and phi = (p - 1) \* (q - 1), d = modinv(e, phi) = 0xbef4bcf3c612ba1

**Task 3**

**Weak key problem caused by Ps and Qs**

The security of RSA is predicated upon the difficulty of factoring Ps and Qs. While it is fast to multiply prime factors P and Q into N, it should be almost impossible to divide N into their original prime factors. This can be achieved by having huge N’s where their respective Ps and Qs would take exponentially long to factor.

However, even with long Ns, patterns resulting from a lack of entropy can result in the P or Q of one N being identical to another. This is when an attacker can deduce the P or Q of long Ns, by finding their GCD, which is significantly faster than completely factoring the same Ns. If GCD =! 1, then this implies that a coprime exists, which could be the common prime factor between Ns. (Note that the existence of coprime could imply other scenarios, but that’s beyond the scope of this project)

The attacker can then divide that common divisor (let’s call it P) by one of the Ns, to retrieve Q. Once the P and Q are retrieved, we can compute phi = (p - 1) \* (q - 1). Since e is public, and we have Phi, a modular inverse similar to Task 2 can get us d, the private key. Voila, we have cracked the RSA!

**Simple description about steps to get the private key.**

**1)** We begin by exploiting the lack of entropy in Task 3’s keys by finding the GCD between n1 and n2. The result will yield Waldo’s Public Key n2 which shares the same common factor P with our own Public Key n1.

**2)** Knowing our common factor P, we can deduce Q = n1 / P. This effectively gives us P and Q, and therefore

Phi = (P - 1) \* (Q - 1).

**3)** As we now have n1, e, and phi, a simple modular inverse function, identical to the one used in Task 2, will yield us our private key! The most critical concept is: P\*Q1=n1 and P\*Q2=n2, therefore P = gcd(n1,n2) if it !=1