6035 project 3 report
ID: tyan37

Task2:

First, I wrote a function called get_factors() to find the prime factors of n. At first, I used brutal force method to find the primes, but it took a long time. After reading the posts online, I start finding prime factor from the $\sqrt[2]{n}$, and decrease the number by 1 in every loop. This method improves the efficiency dramatically.

After getting p and q, I used Euclid's algorithm to calculate modular inverse. Recursive function is used to implement Euclid's algorithm.
Euclid algorithm is referred here:
https://en.wikibooks.org/wiki/Algorithm_Implementation/Mathematics/Extended_Euclidean_algorithm .

Task 3:

When p and q are unknown, the most efficient known method to calculate the private key is to factor N into p and q and use the equation d = e-1 mod (p-1)(q-1) to calculate d. Directly calculating p and q using get_factors() function took a long time. However, calculating greatest common divisor (GCD) is very fast. Distinct moduli that share exactly one prime factor will result in public keys that appear distinct, but the attacker can easily factor both moduli by computing their GCD, p , and dividing to find q1 and q2 . The attacker can then compute both private keys as explained above.

Since Waldo and own key share the same prime factor, we can calculate the greatest common divisor(GCD) of classmate n2 and my own n1. If the GCD is not 1, which means they share a same prime factor, and the classmate is waldo.

GCD is the shared prime factor, we can get the other prime factor by simply using n1/GCD. In this way, we get p and q efficiently. Then we use the same Euclid's algorithm as in task2 to get the private key.

Task4:
From task 4 we know the same message was encrypted with three different RSA public keys, and they have the same e = 3. We can get those equations:
$C_1 = m^3 \bmod N_1$
$C_2 = m^3 \bmod N_2$
$C_3 = m^3 \bmod N_3$

Since m is much smaller than N1 N2 and N3..And I also checked the Jason file and found that all the C0, C1, C2 are the same, Which means the N1, N2, N3 are larger than m^3. Then m = $\sqrt[3]{C}$. So when the e is not large, and N is larger than m^e, attacker can easily get the m just by getting cube root of C.

The trick part is we need to pay attention to precision issue when we calculate. In the recover_mg () function, I made precision to 1000, and also round up the result after get the root result.