Math 25 – Group Programming Assignment 2i – Solutions

1. Let p, q be distinct Sophie Germain primes and let $n = (2p+1)(2q+1)$.

   (a) Prove that $\mathbb{Z}/n\mathbb{Z}^\times$ is not cyclic. That is, there is no primitive element.

   (b) Prove that there exists an element of order $2pq$.

   (c) Let $p = 100811$ and $q = 122231$. Find an element modulo n of the largest possible order.

   Solution. (a) Notice that $2p+1, 2q+1 \geq 5$ are distinct odd primes (by the definition of Sophie Germain). Thus n is not of the form $1, 2, 4, p^e, 2p^e$, so $\mathbb{Z}/n\mathbb{Z}^\times$ does not have a primitive element.

   (b) Both $\mathbb{Z}/(2p+1)\mathbb{Z}^\times$ and $\mathbb{Z}/(2q+1)\mathbb{Z}^\times$ have primitive elements. We choose two such elements α, β (respectively).

   The orders of α and β are $2p$ and $2q$, respectively. By the CRT, we may find a solution x to the system

   $$\begin{aligned} x &\equiv \alpha \pmod{2p+1} \\ x &\equiv \beta \pmod{2q+1} \end{aligned}$$

   Notice for all $m \in \mathbb{N}$ that

   $$\begin{aligned} x^m &\equiv \alpha^m \pmod{2p+1} \\ x^m &\equiv \beta^m \pmod{2q+1} \end{aligned}$$

   Because the CRT lifting map is a bijection, we have that $x^m \equiv 1 \pmod{n}$ if and only if $\alpha^m \equiv 1 \pmod{2p+1}$ and $\beta^m \equiv 1 \pmod{2q+1}$. The smallest such m must be a multiple of $2p$ and $2q$. The least common multiple is $2pq$, and this value is seen to do the trick. Thus, x is an element of order $2pq$.

   (c) Trying elements randomly and checking if they are primitive is pretty efficient. The following code shows 123 is an element of order $2pq$.

```
p = 100811; q = 122231; n = (2*p + 1) * (2*q + 1);

def check_order(x):
    return pow(x, 2*q, n) != 1 and pow(x, 2*p, n) != 1 and pow(x, p*q, n) != 1

assert check_order(123)
```

   So why does it work? A combination of two things. First:

   Lemma 1. *The order of any element modulo n divides $2pq$.*

   Proof. By the CRT, we have $x^m \equiv 1 \pmod{n}$ if and only if $x^m \equiv 1 \pmod{2p+1}$ and $x^m \equiv 1 \pmod{2q+1}$. For any $x \in \mathbb{Z}/n\mathbb{Z}^\times$, we see that $x^{2pq} \equiv 1 \pmod{n}$ by the last statement and Fermat's little Theorem. Thus $m \mid 2pq$. □

   The second is a primitive element test like result. The only possible orders are

   $$1, 2, p, q, 2p, 2q, pq, 2pq.$$

   All we have to do is ensure that the order is not a divisor of $2p$, $2q$, or pq. □

2. A company makes and sells devices. What these devices do is not important for us to know, but what is important is that each device comes with RSA keys. The private key is safely kept hidden and the public key is available for access on the internet.

   It was Bob's job to write the code that generated the RSA keys for these devices. He came up with this

```
def make_rsa_key():
    # load secret_list_of_primes, a list of 50 random 512 bit primes.
    p = random(secret_list_of_primes)
    q = random(secret_list_of_primes)
    assert p != q

    return p*q, rsa_exponents(p, q)
```

   Perhaps Bob had a lot on his mind that day – a mild slight from a stranger that weighs on one's mind more than it should, or, some trouble remembering that eleventh item on the list of groceries he promised to procure for his dear Linda; advil..rosemary... Alas, it does not matter to us – our concern is with the terrible mistake Bob has made.

   Let us assume that `rsa_exponents` does what it is supposed to. Available on canvas is a data file called `pub_keys`. It contains a list of 250 RSA moduli generated with Bob's code.

   Your task: Determine Bob's secret list of 50 primes.

   Solution. Factoring the RSA moduli is hopeless. Instead, we weaponize the birthday paradox. It is a statistical certainty that the RSA moduli will share primes in common. Computing GCDs is extremely fast, so we can just compute the GCDs of all possible pairs and be reasonably confident this will work.

```
with open("pub_keys", "r") as F:
    moduli = [int(x) for x in F.readlines()]

pub_gcds = {gcd(a, b) for a in moduli for b in moduli if a != b}
pub_gcds.remove(1)

assert len(pub_gcds) == 50 # These are Bob's primes.
```

   Note: If N and M are non-equal RSA moduli with a non-trivial gcd, then there exist primes p, q, r such that $N = pq$ and $M = pr$. From Assignment 3, $\gcd(N, M) = p$.

   Remark: Actually, 250 moduli is overkill. For those who've run into Graph Theory, you make the following graph G whose vertices are the primes in Bob's list and where $p \sim q$ if $N = pq$ is one of the published moduli. To break all the keys, you only need each connected component of G to have size at least 3. You need at least 34 moduli to break everything. After this point, the probability you cannot break everything decays exponentially. However, this is an exercise for another course.

   Remark: This type of attack is *very real*. Please math responsibly.

   `https://www.researchgate.net/publication/266352987_Ron_was_wrong_Whit_is_right`

   □