Mathematics 3159A Introduction to Cryptography

Group Assignment [3] Due Date: Oct 27, 2020

Report

Group members.

- Manager: Peter Beaulieu, pbeauli2@uwo.ca
- Programmer: Jinho Kim, jkim2492@uwo.ca
- Reporter: N/A
- Scribe: Dat Vo, dvo9@uwo.ca
- Secretary: N/A

List all group members with their roles and email addresses.

Group meetings. The group met October 24, 2020 at 4:00pm. Setup Google Doc and LaTeX document, and discussed general goals. Meeting was closed at 4:15pm by the host.

Timeline. Meetings were scheduled for 4pm on Saturday and Sunday. Question 1 was completed on Sunday at 5pm and Question 2 was completed on Monday at 2:30pm. The program was completed on Sunday, then added to complete the report.

Alternative approaches. Originally, a static a variable was used to test but then some outputs were not generated properly so a list of possible a values was generated to compensate for this, thus leading to our solution.

Problem

We wish to show that if the decryption exponent can be computed in an RSA-style exchange, then the RSA modulus, $N = pq$ can be factored. This effectively shows that extracting the decryption exponent given the public variables (N, e) is harder than factoring N, which we assume to be infeasible.

Solution

Problem 1.1

1. Pick an integer $1 < a < n$ such that $\gcd(a, N) = 1$. This is fairly simple since N only has two factors.
2. Set $k = 1$ and repeatedly increase k by 1 until 2^k does not divide $de - 1$.
3. Set $m = (de - 1)/2^k$. Going forward, let $a_i = a^{2^i m} \mod N$.
4. Compute and check if $a_0 \equiv 1 \mod N$. If so, then return to step 1 and pick a different a.
5. For $i = 0, 1, ..., k - 1$, compute and check if $a_i \equiv -1$. If so, then return to step 1 and pick a different a.
6. Find the least integer $1 \leq j \leq k$ satisfying $a^{2^j m} \equiv 1$. This integer always exists since $j = k$ implies $a^{2^j m} \equiv a^{de-1} \equiv 1 \mod N$.
7. Return $p = \gcd(a_{j-1} - 1, N)$ and $q = \gcd(a_{j-1} + 1, N)$.

Problem 1.2

This algorithm obviously terminates since there are at most $n - 2$ possible values of a.

Given (N, e, d), and $m = (de - 1)/2^k$ as seen above, this algorithm relies on finding an integer $1 < a < n - 1$ satisfying both

- $a^m \not\equiv 1 \mod n$
- $a^{2^i m} \not\equiv -1 \mod n$ for all $i = 0, 1, ..., k - 1$.

Steps $4, 5, 6$ ensure that such conditions are met. We claim that if a satisfies the conditions, then $\gcd(a^{2^j m} - 1, N)$ is a factor of N where j is as described in step 6.

Proof. Due to step 4, we see that j cannot be 0. Hence $j - 1$ is greater than zero. And due to steps 4,5, a_{j-1} is not 1 or -1. We see that

$$(a_{j-1})^2 \equiv (a^{2^{(j-1)}m})^2 \equiv a^{2^j m} \equiv 1 \mod N.$$

Since a_{j-1} is a non-trivial square root of 1 and we have that $a_{j-1} \pm 1$ are non-zero and satisfy

$$(a_{j-1} - 1)(a_{j-1} + 1) \equiv 0 \mod N.$$

Hence $a_{j-1} \pm 1 \mod N$ are positive integers less than N that p and q divide. Since neither of $a_{j-1} \pm 1$ equal N, it follows that if p divides $a_{j-1} + (-1)^s$ for some s, then q must divide $a_{j-1} + (-1)^{s-1}$. $\square$

Program

```
from gi import generate_input as gi
def check(a, m, n, k):
        #List of a_j
        res = [pow(a, m, n)]

        #Step 4,5
        if res[0] == 1 or res [0] == n-1:
                return -1

        #Step 5
        for i in range(k):
                res.append(pow(res[-1], 2, n))
                if res[-1] == n - 1:
                        return -1
        res.append(1)

        #Step 6
        return res[res.index(1) - 1]
def mr(n, e, d):
        ed = e * d

        #Step 2
        k = 1
        while (ed - 1) % pow(2, k) == 0:
        k = k + 1
                k = k - 1

        #Step 3
        m = (ed - 1) // pow(2, k)

        for a in range(2, n):
                x = check(a,m,n,k)
                if x >= 0:
                        #Step 7
                        return (gcd (n,x-1),gcd (n,x+1))
        return "failed"
def gcd(a, b):
        if (b == 0):
                return a
        else:
                return gcd(b, a % b)
```

Outputs

```
>>solve(1050809297549059495397, 1320206047602447099947, 1269086219575780968883)
(32416189877, 32416187761)

>>solve(1050809291778978996259, 1660290293467749558831, 5916558169245082712171)
(32416188859, 32416188601)

>>solve(1050809307598078897051, 6892753006115176386173, 5429843247213037732333)
(32416189499, 32416188449)

>>solve(1050809311423189007233, 2792254691815120372811, 1328234985524942259377)
(32416188349, 32416189717)

>>solve(1050809342672395878581, 7175469475995114834733, 2798396368228170924011)
(32416189753, 32416189277)

>>solve(1050809251842234639827, 9528383882625167913011, 9123600159162978999011)
(32416187627, 32416188601)

>>solve(1050809321472207649103, 9209391356776123841711, 3457641268863492429311)
(32416189859, 32416188517)

>>solve(1050809275570884406799, 4453523699547889686677, 6990963645637560920833)
(32416189031, 32416187929)

>>solve(1050809269217311777699, 2702818736297378853411, 6476314544619572018933)
(32416188367, 32416188397)

>>solve(1050809294177776720189, 3087086939944407405299, 2584319324939613223211)
(32416189277, 32416188257)
```