Mathematics 3159A Introduction to Cryptography

Group Assignment [4] Due Date: November 24, 2020

Report

Group members.

- Manager: Kexuan Zhao, `kzhao45@uwo.ca`
- Programmer: Dat Vo, `dvo9@uwo.ca`
- Reporter: Nirmal Vettiankal, `nvettian@uwo.ca`
- Scribe: Terry Zeng, `tzeng23@uwo.ca`
- Secretary: Mahima Siali, `msiali@uwo.ca`

Group meetings. The group met for the first time November 20, 2020 at 3pm EST. The meeting ended at 3:41 EST. The second meeting was on November 21, 2020 at 1pm EST. The meeting ended at 2:45 EST. All members were present for both meetings.

Timeline. The first meeting was scheduled November 19, 2020 through a Facebook messenger chat, created on the same day. All subsequent meetings were scheduled in the meeting before it. In the first meeting, a possible solution was discussed. An Overleaf document was created so that the team could work together on the submission. In the second meeting, we worked on coding the proposed solution. Work was done in coderpad.io. The program was written November 20-21, 2020. The meeting ended shortly after the code was completed and everyone understood the solution. We decided that no more meetings were necessary and proof reading/finishing up the document could be done through the messenger chat that was created.

Alternative approaches. There were no alternative approaches that were considered, however there were considerations made for branching off the same approach. For Part 1 Problem 1 it was known that the same random r would generate the same S_1 and S_1' however we considered if it was only the case that $S_1 = S_1'$ if they were created by the same r.

Problem

In Part 1, we were tasked with examining the choice of element r in the ElGamal Digital Signature Algorithm. With A being Samantha's verification key, D and D' being two documents she signed using r, the signatures that she produced were (S_1, S_2) on D and (S'_1, S'_2) on D'.

For Part 1 Problem 1, we were given the verification key A and two signatures (S_1, S_2) and (S'_1, S'_2). We had to come up with an algorithm that determines whether or not they were produced using the same element r.

For Part 1 Problem 2, we had to come up with an algorithm that finds the secret signing key a, if (S_1, S_2) and (S'_1, S'_2) were in fact produced using the same r.

In Part 2, we were tasked with writing a Python function based on the algorithms we created in Part 1. The function "solve" was to take the following inputs:

- p, a large prime;
- g, belonging to $\mathbb{F}_p^*$;
- A, congruent to $g^a \bmod p$;
- (S_1, S_2) and (S'_1, S'_2), where both pairs are valid signatures for D and D', produced using the secret signing key a.

And return first whether $(S_1, S-2)$ and (S'_1, S'_2) were signed using the same element r, and then output the secret signing key a if they were signed by the same r.

We then ran our Python function on the `generate_input` file and recorded our outputs.

Solution

1.1: **Claim**: $S_1 = S'_1$ if and only if Samantha uses the same element r to create both signatures.

Proof First we show that reverse direction, i.e if Samantha uses the same element r to sign two documents, we will see $S_1 = S'_1$.
We recall that

$$S_1 \equiv g^r \pmod p$$

and

$$S'_1 \equiv g^{r'} \pmod p$$

. Therefore, if $r = r'$, we find that $S_1 = S'_1$.
We are now left with showing that $S_1 = S'_1$ ONLY when Samantha chooses the same r. We recall we have g, a primitive root of p. As g^i generates all elements of p for $i = 0, 1, \ldots, p-1$.

Then by definition, we see that for $r \neq r'$, we will have $g^r \pmod p \neq g^{r'} \pmod p$. Therefore, if $S_1 = S_1'$ for $S_1 = g^r \pmod p$ and $S_1' = g^{r'} \pmod p$, we must see that $g^r \pmod p = g^{r'} \pmod p$, and therefore $r = r'$. Therefore, if we find that $S_1 = S_1'$, the two signatures must have been generated with the same element r.

Thus, it suffices to show that $S_1 = S_1'$ to show that Samantha used the same element r to sign both documents, as will be outlined in the algorithm below.

Algorithm Check if $S_1 = S_1'$. If they are equal, then the random element is the same.

1.2 If (S_1, S_2) and (S_1', S_2') were determined to be produced by the same random element in (1.1), we can proceed to find the secret signing key a. Because $S_1 = S_1'$, we have

$$aS_1 + rS_2 \equiv D \pmod{(p-1)} \tag{1}$$

$$aS_1 + rS_2' \equiv D' \pmod{(p-1)} \tag{2}$$

So multiplying S_2' by (1) and subtracting S_2 multiplied by (2) we have

$$S_1(S_2' - S_2)a \equiv S_2'D - S_2D' \pmod{(p-1)}$$

we know $S_1(S_2' - D_2)$ and $S_2'D - S_2D'$, then we can solve the congruence if $gcd(S_1(S_2' - D_2), p-1) = 1$ with a unique a. If $gcd(S_1(S_2' - D_2), p-1) \neq 1$, we get multiple solutions and check for $g^a \equiv A \pmod p$.

Algorithm
Input. $(p, g, A, D, D', S_1, S_2, S_1', S_2')$
Output. secret signing key a

```
1. Compare $S_1$ and $S_1'$, if $S_1 \neq S_1'$, return "no".
2. Let $A \equiv S_1(S_2' - S_2) \pmod{(p-1)}$
   $B \equiv S_2'D - S_2D' \pmod{(p-1)}$
   $d = gcd(a, p-1)$
3. if $d = 1$, $a \equiv A^{-1}B \pmod{(p-1)}$, return a.
4. else, let $m = \frac{p-1}{d}$.
   $a \equiv A^{-1}B \pmod{(p-1)}$
5. loop $i = 1, 2, ..., d$
   6. If $g^a \equiv A \pmod p$, return $a$
   7. else, $a \equiv a + m \pmod{(p-1)}$
```

Program

```python
def gcd(a, b):
    while b:
        a, b = b, a % b
    return a


def inverse(n: int, a: int):
    t, newt = 0, 1
    r, newr = n, a

    while newr != 0:
        quotient = r // newr
        t, newt = newt, t - quotient * newt
        r, newr = newr, r - quotient * newr

    if r > 1:
        raise ArithmeticError(f"{a} is not invertible mod {n}")
    if t < 0:
        t = t + n

    return t


def solve(p, g, A, D, Dp, S1, S2, S1p, S2p):
    if S1 != S1p:
        return "no"

    a = (S1 * (S2p - S2)) % (p - 1)
    b = ((S2p * D) - (S2 * Dp)) % (p - 1)
    d = gcd(a, p - 1)
    if d == 1:
        ans = (inverse(p - 1, a) * b) % (p - 1)
        if pow(g, ans, p) == A:
            return ans
        else:
            return "failed"
    else:
        a //= d
        b //= d
        m = (p - 1) // d
        ans = (inverse(m, a) * b) % m
        for i in range(d):
            if pow(g, ans, p) == A:
                return ans
            ans = (ans + m) % (p - 1)
        return "failed"
```

Output

```
» solve(33555751, 243, 30542048, 19, 14, 295845, 20826997, 14325870, 33485282)
no
» solve(33554699, 32, 32768, 14, 13, 2043221, 31335333, 2043221, 32352142)
3
» solve(33555553, 16807, 14030825, 8, 6, 11030354, 28131596, 11478229, 14555684)
no
» solve(33555617, 243, 14348907, 10, 5, 13845724, 12977014, 30017090, 26275185)
no
» solve(33555817, 3125, 9765625, 8, 23, 7523293, 28614582, 22256958, 3172621)
no
» solve(33555493, 32, 32768, 10, 3, 11152483, 15342455, 29806492, 13155273)
no
» solve(33555383, 16807, 11701571, 11, 20, 447397, 16228604, 28817193, 13965411)
no
» solve(33554579, 32, 33554432, 31, 16, 24539544, 31870077, 24539544, 22718828)
5
» solve(33555659, 32, 1048576, 17, 10, 1105213, 14101769, 18155557, 5893352)
no
» solve(33555817, 3125, 9765625, 8, 23, 7523293, 28614582, 22256958, 3172621)
no
```