Mathematics 3159A Introduction to Cryptography

Group Assignment 4 Due Date: Nov 24th, 2020

Problem

Given a verification key A and two signatures $(S_1, S_2), (S_1^{'}, S_2^{'})$, determine if the random element $r = r^{'}$. And if $r = r^{'}$, determine the secret signing key a.

Solution

1.1

Input: Signatures (S_1, S_2) and $(S_1^{'}, S_2^{'})$.
Output: $True$ if $r = r^{'}, False$ otherwise.
Step 1: If $S_1 = S_1^{'}$ return $True$
Step 2: Else return $False$.

Correctness:
Since $S_1 \equiv g^r \pmod p$ and $S_1^{'} \equiv g^{r^{'}} \pmod p$, we conclude that if $S_1 = S_1^{'}$ then $r = r^{'}$.

1.2

Input: prime p, primitive root g, signatures (S_1, S_2), $(S_1^{'}, S_2^{'})$ and documents D and $D^{'}$.
Output: secret key a
Step 1: Compute $X = S_1(S_2^{'} - S_2)$
Step 2: Compute $Y = S_2^{'}D - S_2D^{'}$
Step 3: Compute $i = gcd(X, p-1)$
Step 4: if $i = 1$, solve $Xa \equiv Y \pmod{p-1}$ for a, return a
Step 5: else solve $Xa \equiv Y \pmod{p-1}$ for $a = \{a_1, a_2..., a_n\}$
Step 6: Solve $S_1 \equiv g^r \pmod p$ for r
Step 7: For $a = a_1, a_2, ...a_n$: Solve $g^a \equiv r^{'} \pmod p$ for $r^{'}$
Step 8: if $r^{'} = r$, return a.

Correctness:
We have

$$S_1 a + S_2 log(S_1) \equiv D \pmod{p-1}$$

and

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

Since $S_1 = S_1^{'}$, we merge two equations, and get:

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

If $gcd(S_1(S_2^{'} - S_2), p-1) = 1$ then $S_1(S_2^{'} - S_2)a \equiv S_2{'}D - S_2D^{'} \pmod{p-1}$ has a unique solution a which is the secret key.
If $gcd(S_1(S_2^{'} - S_2), p-1) > 1$ then there are multiple solutions. Then we compute all the solutions $a = a_1, a_2..., a_n$, we substitute $a_1...a_n$ and solve $g^a \equiv r^{'} \pmod{p}$ for r.
If $r^{'}$ is equal to the random element r, then the associated a is the secret key.

Program

```
### Bradley Assaly-Nesrallah
### bassalyn@uwo.ca
### 250779140
### Written Oct 22, 2020
### Usage: solve(p,g,A,D,D',S1,S2,S'1,S'2)

from generate_input import generate_input

##Implementation of fast powering algorithm
##input:base, power and modulo p integers
##output base^power mod p integer
def rapidExponentiation( base, power, p ):

    result = 1 #start with 1
    while power != 0: # loop until power = 0
        if power % 2 == 1: #odd pow -1 exp
            result = (result * base) % p
        power = power // 2 # ^power = ^power/2
        base = (base * base) % p # base = base * base

    return result

##implementation of extended euclidean algorithm go compute gcd
##input integers a and b
##output gcd of the integers a and b
def eeagcd(a, b):
    x,y, u,v = 0,1, 1,0 ##base case
    while a != 0: ##while a not 0 loop
        q, r = b//a, b%a ##EEA implentation from textbook
        m, n = x-u*q, y-v*q
        b,a, x,y, u,v = a,r, u,v, m,n
    gcd = b
```

```
    return gcd ##eturns gcd


#compute gcd recursively and outputs a tuple
def gcd(a, b):
    if a == 0:
        return (b, 0, 1)
    else:
        g, y, x = gcd(b % a, a)
        return (g, x - (b // a) * y, y)

##implementation of extended euclidean algorithm go compute inverse
##input integers a and b
##output no if gcd(a,b) is not 1, inverse of a otherwise

def eeainv(a, m):
    g, x, y = gcd(a, m)
    if g != 1:
        raise Exception('modular␣inverse␣does␣not␣exist')
    else:
        return x % m


# Based on math proof of part 1
# solve function to obtain the secret key a, if possible
# Input: (p,g,a,d,dprime,s1,s2,sprime1,sprime2)
# p prime, g is primitive root mod p, A is form g^a mod p,
# (s1,s2) and (sprime1,sprime2) are two valid signatures for docs D D' respect
# Output:
# returns no if (s1,s2) and (sprime1,sprime2) were produced using diff random elements
# returns the secret signing key a if produced using same ramdom element
def solve(p,g,a,d,dprime,s1,s2,sprime1,sprime2):
    if s1 != sprime1: ##check if same produced same element
        return "no"
    B = (s1*(sprime2-s2))%(p-1) ## compute X and Y from algorithm
    C = (sprime2*d - s2*dprime)%(p-1)
    gcd = eeagcd(B,p-1) ## get gcd using eea
    Q = B//gcd ## divide through by the gcd
    R = C//gcd
    N = (p-1)//gcd
    Qinv = eeainv(Q,N) ## compute the inverse using EEA
    if gcd==1:
        return (Qinv * R)%N
    s= [(Qinv * R) % N] ## iterate though s solutions
```

```
    for i in range (1,gcd):
        s.append(((s[i-1]+N)%(p-1))) ## populate s values
    for i in range(0,gcd):
        if rapidExponentiation(g, s[i], p) == a: ##if g^s = a then return a
            return s[i]
    return -1


def check(p,g,a,d,dprime,s1,s2,sprime1,sprime2):
    i = solve(p,g,a,d,dprime,s1,s2,sprime1,sprime2)
    if i == "no" or i == -1:
        return "true"
    if (rapidExponentiation(g,i,p))==a :
        return "true"
    else:
        return "false"

generate_input(140)
input_tuples = [
    (33555913, 3125, 29098177, 25, 24, 15569550, 2569481, 21321641, 11501461),
    (33554467, 32, 33554432, 27, 7, 18170831, 5073760, 18170831, 24247740),
    (33555449, 243, 14348907, 10, 21, 17586366, 30939856, 26510018, 6373521),
    (33555527, 3125, 12514351, 31, 1, 12466151, 13363702, 30597607, 6454756),
    (33554891, 32, 1048576, 19, 10, 14962679, 8354491, 14962679, 26657158),
    (33555341, 243, 14348907, 10, 3, 26274757, 33200383, 27975835, 32908014),
    (33555883, 32, 33554432, 28, 17, 4985776, 5040184, 14126137, 21725694),
    (33554771, 32, 32768, 9, 30, 3525560, 14931663, 3525560, 11881230),
    (33555583, 7776, 7707241, 29, 24, 19706746, 24029217, 15187996, 25616120),
    (33555281, 7776, 20987321, 19, 16, 6220013, 25627651, 13348745, 6911068)
]

##main function, solves for all tuples obtained from generate input.py
if __name__ == "__main__":
    for tuple in input_tuples: ##solves and outputs for each tuple
        print(
            'solve({0},␣{1},␣{2},␣{3},␣{4},␣{5},␣{6},␣{7},␣{8})'.format(tuple[0],
            tuple[1],tuple[2],tuple[3],tuple[4],tuple[5],tuple[6],tuple[7],
            tuple[8]),
            solve(tuple[0], tuple[1], tuple[2],tuple[3],tuple[4],tuple[5],tuple[6],
            tuple[7],tuple[8]),
            check(tuple[0], tuple[1], tuple[2], tuple[3], tuple[4], tuple[5],
            tuple[6], tuple[7], tuple[8]),
            sep='\n\t\t=',
            end='\n\n'
```

)

Output

```
solve(33555913, 3125, 29098177, 25, 24, 15569550, 2569481, 21321641, 11501461)
        =no
        =true

solve(33554467, 32, 33554432, 27, 7, 18170831, 5073760, 18170831, 24247740)
        =5
        =true

solve(33555449, 243, 14348907, 10, 21, 17586366, 30939856, 26510018, 6373521)
        =no
        =true

solve(33555527, 3125, 12514351, 31, 1, 12466151, 13363702, 30597607, 6454756)
        =no
        =true

solve(33554891, 32, 1048576, 19, 10, 14962679, 8354491, 14962679, 26657158)
        =4
        =true

solve(33555341, 243, 14348907, 10, 3, 26274757, 33200383, 27975835, 32908014)
        =no
        =true

solve(33555883, 32, 33554432, 28, 17, 4985776, 5040184, 14126137, 21725694)
        =no
        =true

solve(33554771, 32, 32768, 9, 30, 3525560, 14931663, 3525560, 11881230)
        =3
        =true

solve(33555583, 7776, 7707241, 29, 24, 19706746, 24029217, 15187996, 25616120)
        =no
        =true

solve(33555281, 7776, 20987321, 19, 16, 6220013, 25627651, 13348745, 6911068)
        =no
        =true
```