EE 720
An Introduction to Number theory and Cryptography

Team Members:

Niklesh Lalwani (Team Leader)
Rohan Jain
Utkarsh Sharma
Shardul Jade
Rushabh Kapasi
Praful Bhansali
Sahil Agarwal
Pradeep Yadav
Balagopal K S
Priyansh Tiwari

PROBLEM STATEMENT:

Discrete log computation over F_p^* by Baby Step Giant Step algorithm using Pohling Hellman reduction step.

ABSTRACT:

If p is a prime and g and y integers, then computation of x such that

$$y = g^x \bmod p$$

is called discrete logarithm problem.

Often, g is taken to be the primitive root mod p, which means that every y is a power of g. If g is not a primitive root, then discrete logarithm will not be defined for some values of y.

The security of many public key cryptosystems is based on the difficulty of this problem.

IMPLEMENTATION:

1) Reduction using Pohling Hellman: Problem of computing the discrete logarithm in a cyclic group G can be reduced to a discrete logarithm problem in a cyclic group of prime power order.
2) BSGS: Once the problem is reduced to computing discrete logarithm in a cyclic group of prime power order, we use Baby Step Giant Step algorithm to compute this.
3) Chinese remainder theorem: After using BSGS to compute discrete logarithms for prime power order cyclic groups, we finally use CRT to get the final answer.

CODE:

```
#alpha=gamma^x
def fun(alpha,gamma, prime,p,e):
   x=[]
   gt=pow(gamma,(prime-1)/p,prime)

   for i in range(1,e+1):
      pi=pow(p,i)
      at=pow(alpha,(prime-1)/pi,prime)

   ###############################################
   #baby step giant step by using hash map
      flag=false
      found=false
      baby=dict()
      baby[at%prime]=0
      m=1+floor(sqrt(p))


      for r in range(0,m):
         tmp=(at*pow(gt,-r,prime))%prime
         baby[tmp]=r
         if(tmp==1):
            x.append(r)
            flag=true
            found=true
            break

      if (flag==false):
         delta=pow(gt,m,prime)
         for q in range(1,m):
             temp=pow(delta,q,prime)
             if(temp in baby):
                x.append(baby[temp]+m*q)
                found=true
                break

      if(found==false):
         return -1
    ###############################################
      c1=((p^(i-1))*x[i-1])
      t=pow(gamma,-c1,prime)
      alpha=(alpha*t)%prime

   ans=0
   for i in range(0,e):
      ans=ans+x[i]*pow(p,i)
```

```
    return ans

def dl(alpha,gamma,prime):
#find factors of (prime-1)
    f=factor(prime-1)
    a=[]
    b=[]

    for i in range(0,len(f)):
        chk=fun(alpha,gamma,prime,f[i][0],f[i][1])
        if(chk==-1):
            continue
        a.append(chk)
        b.append(pow(f[i][0],f[i][1]))

#Compute final answer using Chinese Remainder Theorem
    ans=crt(a,b)
    if(pow(gamma,ans,prime)==(alpha%prime)):
        return ans
    return 'NOT_FOUND'


#*#*#*#*#*#*#*#*#*#*#*#*#*#*#*#*#*#*#*#*#*#*#*#*#*#*#*#*#*#*#*#*#*#*#*#*#
*#*#*#*#*
#dl take arguments alpha,gamma,prime
#alpha=gamma^x (mod prime)
alpha=2804144338
gamma=5
prime=4093082899
factor(prime-1)
#start computing time
t=cputime()
x=dl(alpha, gamma,prime)
x
#show computation time
cputime(t)
#check if computed value is correct, it should come out equal to
pow(alpha, x, prime)
```

Results and Observations:

We computed discrete log for several 10 digit primes and few 20 digit primes. Observations are shown for randomly chosen five 10 digit prime numbers and one 20 digit prime number.

Prime	alpha	gamma	Factors of (prime-1)	Discrete logarithm(x)	Computational time
10 digit primes					
4093082899	2804144338	5	2 * 3 * 23 * 3929 * 7549	983232	0.00537999
9576890767	3636224069	13	2 * 3^3 * 103 * 683 * 2521	234512	0.00510500
3367900313	2096862853	6	2^3 * 7 * 109 * 551753	432512	0.01540800
9848868889	7466883511	12	2^3 * 3 * 31 * 13237727	8649975400	0.06047099
2860486313	156043878	25	2^3 * 31 * 73 * 158003	864975400	0.00964999
20 digit prime					
48112959837082048697	44150717785176697051	5	2^3 * 277 * 10091 * 57467 * 37440323	815911532366213	0.12610000