3^{nd} Assignment Information Security

Francesco Segala 3521885 Manos Gionanidis 3542068

October 3, 2017

Exercise 10

Linear Diophantine equation
spider 8 legs
beetle 6 legs

So because none of them is mutilated it means that the equation is going to be s*8 + b*6 = 56.
Dividing out this common factor gives s*2*4 + b*2*3 = 2*28 so the equation is like this : s*4 + b*3 = 28 .
So we try to assume that the lowest number of spiders/beetles are 0. If This happen we have (suposse b(beetles) = 0) s*4 + 0*3 = 28 So s = 28/4 = 7 . The first pair for this solution is (s,b) = (7,0) but the assginment says that in the box there are box spiders and beetles so this pair is not possible. As tha same for the pair s = 0 first of all because we can not devide exactly the 28 with 3 and second because there are beetles in the box.

Second assumption is that the beetlesNumber>0 and spidersNumber>0. So lets run a code and found the pairs 0< (s,b)<28
The code is only to check the correctness of my assuptions.

```
def ex10():
  for s in range(1,5):
    for b in range(1,9):
      if(s*4 + b*3 == 28):
        print 'Success Pair: (s,b) = (',s,',',b,')'
      else:
        print 'Fail Pair: (s,b) = (',s,',',b,')'
```

Now we have to think about the maximum values that the s,b can take. Let's start with s (fact s<7 because 4*7=28 so b=0) if s=6 means that 4*6=24 and 28-24=4 but we have b*3 so the reminder must be multiple of 3. If s=5 5*4=20 , 28-20=8 8 is not multiple of 3 . If s=4 4*4=16 , 28-16=12 12 is a multiple of 3 so for s the max is 4 (fact s<=4). Now about b if b=9 3*9=27 , 28-27=1 no multiple of 4.If b=8 8*3=24 , 28-24=4 which is multiple of 4 (fact b<=8). We have found the max values so we can use code in order to find the valid combinations starting from tha max of every animal. The valid pairs are (s,b) = (1,8) = (4,4) .

Excercise 11

```
#!usr/bin/python

def ex11():
#compute number = (43210^23456)%99987
#computation take place by squaring and multiplying
base = 43210
power = 23456
modulo = 99987
#make power binary with 20bits
binaryPower = "{0:b}".format(power)
#I skip the first bit because is number^x(and x=1) so number^x=number
baseBase = base
for x in binaryPower[1:]:
    base = anadromic(base,int(x),modulo,baseBase)
print 'Result from hidden slides procedure: ',base
print 'Result using normal operators: ',(43210**23456)%99987

def anadromic(number,x,modulo,baseBase):
#baseBase because i want every time that the bit is 1 to multiple
# with the initial base the following procedure is from the hidden
# slides, (squaring and multiplying)
if(x==0):
    result = (number**2)%modulo
    return result
elif(x==1):
    result = ((number**2)%modulo*baseBase)%modulo
    return result
```

Exercise 12

Using the Schneier's algorithm in python tecnology we found that the largest Generator for 7919 is : 7917 Then all the generators for 23 are: [5, 7, 10, 11, 14, 15, 17, 19, 20, 21]. This is the code for the exercise:

```
    def prime_factors(n):
```

```
    primfac = []
    d = 2
    while d*d <= n:
        while (n % d) == 0:
            primfac.append(d)
            n //= d
        d += 2
    if n > 1:
       primfac.append(n)
    return primfac

def compute_generators(gen,prime,q):
    computed=[]
    for i in q:
        computed.append(gen**((prime-1)/i)%prime)
    return computed


def compute_prime_factors_q(prime):
    p_factors=prime_factors(prime-1)
    computed=[]
    res=[]
    for g in range(2,prime):
        computed=compute_generators(g,prime,p_factors)
        if 1 not in computed:
            res.append(g)
    return res
```

Excercise 13

The procedure is the same like in the hidden slide.

Table 1: CRT

cipher	public factors N	ai	mi	ni	qi
20	493	1	493	487,531	75
382	517	1	517	464,899	156
622	943	1	943	254,881	515

M = m_1 *m_2 * m_3 = 240,352,783 .
X = c_1 * n_1 * q_1 +c_2 * n_2 * q_2 + c_3 *n_3 * q_3 = 110,081,588,438 .
X % M = 13,824 .
m^3 = 13,824 so m=24 .

For the decription I used the CRT and for encryption I used this formula
c = (m^e) mod n .
Public key = (n,e)
This is the source code for en/decryption.
Inputs N1 C1 N2 C2 N3 C3.

```
#!usr/bin/python

import sys
import fractions
import math

#CRT prcedure
def chinese_remainder(n, a):
  sum = 0
  prod = reduce(lambda a, b: a*b, n)

  for n_i, a_i in zip(n, a):
p = prod / n_i
sum += a_i * mul_inv(p, n_i) * p
  return sum % prod


#checking for common factors
def mul_inv(a, b):
  b0 = b
  x0, x1 = 0, 1
  if b == 1: return 1

  while a > 1:
try:
  q = a / b
  a, b = b, a%b
  x0, x1 = x1 - q * x0, x0
except:
  print "Bad N values (check no common factors in N vals)"
  return 0
  if x1 < 0: x1 += b0
  return x1


def GCD(a,b):
#The Euclidean Algorithm
a = abs(a)
```

```
b = abs(b)
while a:
        a, b = b%a, a
return b

def GCD_List(list):
#Finds the GCD of numbers in a list.
#Input: List of numbers you want to find the GCD of
  #E.g. [8, 24, 12]
#Returns: GCD of all numbers
return reduce(GCD, list)

def decrypt():
nval=[]
aval=[]


#initialize chinese remainder theorem input
nval.append(int(N1))
nval.append(int(N2))
nval.append(int(N3))
#nval.append(int(N4))
aval.append(int(C1))
aval.append(int(C2))
aval.append(int(C3))
#aval.append(int(C4))

n = nval
a = aval
g = GCD_List(n)

print "N1: ",N1,'\n'
print "N2: ",N2,'\n'
print "N3: ",N3,'\n'
#print "N4: ",N4,'\n'
print "Cipher1: ",C1,'\n'
print "Cipher2: ",C2,'\n'
print "Cipher3: ",C3,'\n'
#print "Cipher4: ",C4,'\n'
print "e: ",e,'\n'


if (g>1):
  print 'Computing error'
```

```
else:
  result = chinese_remainder(n, a)
  count=0


for str1 in nval:
  print "x mod "+str(str1)+"="+str(aval[count])
  count=count+1

print "Result (x) is: ",result,"\n"
m = 10**(math.log10(result)/int(e))
print 'Calculated value of m is ',int(round(m))


def modinv(a, m):
g, x, y = extended_gcd(a, m)
if g != 1:
  raise ValueError
return x % m

def extended_gcd(aa, bb):
  lastremainder, remainder = abs(aa), abs(bb)
  x, lastx, y, lasty = 0, 1, 1, 0
  while remainder:
      lastremainder, (quotient, remainder) =
                      = remainder, divmod(lastremainder, remainder)
      x, lastx = lastx - quotient*x, x
      y, lasty = lasty - quotient*y, y
  return lastremainder, lastx * (-1 if aa < 0 else 1),
                                         lasty * (-1 if bb < 0 else 1)




def encrypt():
print ('Encrypt with CRT')
#primes for the key generation
p=47# p > q fact
q=29
n = int(raw_input('Give the modulus: '))
#n = p*q with the given primes
print 'Key: ',n
e = int(raw_input('Give the exponent: '))
```

```
print 'Public key for RSA-CRT is : ',n,',',e

m = int(raw_input('Enter the message that you want to encrypt: '))
#We can use the CRT to compute (m = c**d mod n)more efficiently
cstr = ""
#m = int(raw_input("Enter your message m: "))
#for m in [elem.encode("hex") for elem in mstr]:
c = ( m ** e ) % n

print "Your encrypted message m is now a ciphertext c  =", c


def ex13():
choice = raw_input('Write e for encryption and d for decryption: ')
if(choice=='d'):
  decrypt()
elif(choice=='e'):
  encrypt()
else:
  print 'Input error'

ex13()
```

With an encryption exponent of e = 3, the following cube root attack is possible. If the plaintext M satisfies M < iV 1 / 3 , then C = M e = M 3 , that is, the mod N operation has no effect. As a result, an attacker can simply compute the usual cube root of C to obtain M. In practice, this is easily avoided by padding M with enough bits so that, as a number, M > N 1 ' 3 .
N=47*29=1363

Exercise 14

The X-coordinates of the point are $44 * 57 * P1 = 57 * 44 * P1 = (35, 20)$.
Alice sends to Bob the point $44 * (3, 6) = (13, 16)$, Bob sends to Alice $57 * (3, 6) = (7, 8)$.
To compute 57*P1 are required 57 addition , but a simple optimization can be performed so we came up with 12 addition, that for $57 = 2^5 + 2^4 + 2^3 + 2^0$. The details of how we do this is explained in Exercise 17.

Exercise 15

The aim of this exercise was to deal with an ECC DH key exchange. So this is the *secret* i.e m * (Xn, Yn) where Xn and Yn are computed by the T.A.

starting from our public key:
a 27
b 152
N 229
X1 32
Y1 11
Xm 79
Ym 40

The shared secrert is : $(158, 82)$

Excercise 16

This is the plaintext:

When used to define a specific genre or type of film or television programme, drama is usually qualified with additional terms that specify its particular subgenre, such as "political drama," "courtroom drama," "historical drama," "domestic drama," or "comedy-drama." These terms tend to indicate a particular setting or subject-matter, or else they qualify the otherwise serious tone of a drama with elements that encourage a broader range of moods.

Excercise 17

We use an optimization for the multiplication algorithm: instead of repeat n*P as P+P+....+P n times, we use the insight that P+P=2P , storing 2P as P1 we compute P1+P1= 4P and so on, so we compute $\lfloor(log(n))\rfloor$ sums , then we repeat the same iteration for the factor n1=$n - \lfloor(log(n))\rfloor$.
Finally we came up with this solution:

```
p17=(97339010987059066523156133908935, 149670372846169285760682371978898)
a17=32109476812914760189251487282566 8
b17=43078231514021827426227669432319 7
N17=56453825208444155624701690273525 7
n17=48603545970286694910611304838118 2
```

```
def find_expo(n):
    exp=int(log(n,2))
    res=[exp]
    n=n-2**exp
    while n > 1 :
        exp=int(log(n,2))
```

```
        res.append(exp)
        n-=2**exp
    if n == 1 :
        res.append(0)
    return res

def multiplication_op((x,y),n,a,b,N):
    expos=find_expo(n)
    xr,yr=0,"inf"
    res=[0]*len(expos)
    for i in range(len(expos)):
        res[i]=multiply((x,y),2**expos[i],a,b,N)
         #this is a simple multiplication that use the insight stated #
         above for computing power of 2 multipplication
    for i in res:
        xr,yr=add((xr,yr),i,a,b,N)
    return xr, yr

print multiplication_op(p17,n17,a17,b17,N17)
```

and this is the point computed :
(367385334535545015949873084595410L,171995391554293041834290054849881L)