Nirbhay Sharma (B19CSE114)

Cryptography - Assignment-5

Que-1

output considering first 32 bits (4 bytes = 4 chars of hash)

1. x corresponding to emailId(sharma.59@iitj.ac.in) - log(iterations) = 16
2. finding (x, y) - log(iterations) = 9

```
iterations: 10922 ----------
len: 1
m06awl3df@iitj.ac.in

iterations: 2758 ----------
len: 1
l.hg0z97b@iitj.ac.in

iterations: 50298 ----------
len: 1
di6y9j5v4@iitj.ac.in

iterations: 30989 ----------
len: 1
qcmo09by3@iitj.ac.in

iterations: 44514 ----------
len: 1
cxeoj8w45@iitj.ac.in

55417
```

```
iterations: 523 ----------
len: 1
('0mv14jsiq@iitj.ac.in', '013.v9ueo@iitj.ac.in')

iterations: 104 ----------
len: 1
('vqszyxgib@iitj.ac.in', '6yj5gnb7f@iitj.ac.in')

iterations: 161 ----------
len: 1
('43motgvyn@iitj.ac.in', 'xg9pqi31t@iitj.ac.in')

iterations: 367 ----------
len: 1
('3txy25m1d@iitj.ac.in', 'ehv427nlz@iitj.ac.in')

iterations: 345 ----------
len: 1
('52izdcspr@iitj.ac.in', '7qfgjz4d.@iitj.ac.in')

333
```

it is clearly visible that finding an x which has 32 bit common with (sharma.59@iitj.ac.in) is hard as compared to finding two values (x, y) such that their first 32 bits are matching as the first case is the case of second pre image matching and second case is the case for collision and it is always easy or less expensive (computationally) to find collisions than to find second pre image, and hence first case took many iterations to find out the answer.

output considering first 16 bits (2 bytes = 2 chars of hash)

1. x corresponding to emailId(sharma.59@iitj.ac.in) - log(iterations) = 8
2. finding (x, y) - log(iterations) = 5

```
iterations: 55 ----------
len: 1
c1awylu7z@iitj.ac.in

iterations: 290 ----------
len: 1
ve630l1ub@iitj.ac.in

iterations: 46 ----------
len: 1
zkgxuv0w.@iitj.ac.in

iterations: 18 ----------
len: 1
wi6m7t2ds@iitj.ac.in

210
```

```
iterations: 39 ----------
len: 1
('yg.8lv93w@iitj.ac.in', 'y3xwv6kco@iitj.ac.in')

iterations: 43 ----------
len: 1
('vji7k3g80@iitj.ac.in', '3caxep5lf@iitj.ac.in')

iterations: 12 ----------
len: 1
('denq3hz8.@iitj.ac.in', '1a.0fmjtc@iitj.ac.in')

iterations: 33 ----------
len: 1
('yed4ub0xc@iitj.ac.in', 'enwyua94z@iitj.ac.in')

iterations: 16 ----------
len: 1
('uhcv196s4@iitj.ac.in', 'g0.7z681h@iitj.ac.in')

23
```

it is clearly visible that finding an x which has 16 bit common with (sharma.59@iitj.ac.in) is hard as compared to finding two values (x, y) such that their first 32 bits are matching as the first case is the case of second pre image matching and second case is the case for collision and it is always easy or less expensive (computationally) to find collisions than to find second pre image, and hence first case took many iterations to find out the answer.

Que-2

output of the code for RSA encryption and decryption

```
sharma406@LAPTOP-N4BIN1J0:/mnt/d/coding assn sem6/cryptography-assn/assn5$ python3 q2_b19cse114.py
P:  747901531573246085358591762583193586680306334902816940811973003291716112619321759790385261930999251747590499637382667180859521
3874349109354048353
Q:  72182997546261400134527883647728782272344416817336711407362032464061018205300137331671141145015760784218913100743869934920226
5685478866994031023
private d: 10797154883679353051932038494943003358135277860405235893906167596348491411926963087912605238689426214925894347137837167
38123102479367698308172575228722753020922827601545641962672755096886708409243673926701156876407447013997951136523645588274008015473
19496082147717050132048318824655035195149
public n:  53985774418396765259660192474715016790676389302026179469530837981742457059634815439563026193447131074629471735689185838
906155123968384915408628776133452721400742247115988088738451850109410592477548918268561419398085433162087269031822587015201653954
5119198871340615439886115395125152405511
public e: 5
####################### original_text #######################
Nirbhay Sharma is a good boy !!! and he studies at IIT Jodhpur and he likes to do exercise and yoga
####################### encrypted text #######################
0xac16c8e0 0x2f8b91c89 0x47ba2f320 0x21ac75e20 0x2d52e8000 0x1ffd869e1 0x609fdb0d9 0x2000000 0xeac8fd83 0x2d52e8000 0x1ffd869e1 0x
 0x39517623d 0x1ffd869e1 0x2000000 0x2f8b91c89 0x4aedcc023 0x2000000 0x1ffd869e1 0x2000000 0x2b2fb2f87 0x3ec5f782f 0x3ec5f782f 0x2
0x2000000 0x21ac75e20 0x3ec5f782f 0x609fdb0d9 0x2000000 0x25528a1 0x25528a1 0x25528a1 0x2000000 0x1ffd869e1 0x3bff052e0 0x2540be40
00 0x2d52e8000 0x272736815 0x2000000 0x4aedcc023 0x4e3e6b400 0x51acd0565 0x2540be400 0x2f8b91c89 0x272736815 0x4aedcc023 0x2000000
9e1 0x4e3e6b400 0x2000000 0x7b908fe9 0x7b908fe9 0xf9461400 0x2000000 0x84435aa0 0x3ec5f782f 0x2540be400 0x2d52e8000 0x41a700000 0x
 0x47ba2f320 0x2000000 0x1ffd869e1 0x3bff052e0 0x2540be400 0x2000000 0x2d52e8000 0x272736815 0x2000000 0x36bc9ac00 0x2f8b91c89 0x3
0x272736815 0x4aedcc023 0x2000000 0x4e3e6b400 0x3ec5f782f 0x2000000 0x2540be400 0x3ec5f782f 0x2000000 0x272736815 0x5cb278000 0x27
x47ba2f320 0x236d590d3 0x2f8b91c89 0x4aedcc023 0x272736815 0x2000000 0x1ffd869e1 0x3bff052e0 0x2540be400 0x2000000 0x609fdb0d9 0x3
0x2b2fb2f87 0x1ffd869e1
####################### decrypted text #######################
Nirbhay Sharma is a good boy !!! and he studies at IIT Jodhpur and he likes to do exercise and yoga
```

Que-3

1. proving that a cyclic group of n elements has $\phi(n)$ generators

output of the code for sophie_germain_prime and diffie_hilmann_exchange

- according to definition sophie germain prime is a prime number p such that $2p + 1$ is also a prime (safe prime)

```
sharma406@LAPTOP-N4BIN1J0:/mnt/d/coding assn sem6/cryptography-assn/assn5$ python3 q3_b19cse114.py
sphie_germain_prime:   13344560613885683016956070482243112356334510034328720352835313621629930682301702354555
99216602853214359001820521861200842232810722006781290631188685079839
found generator:  929623431195408500820249953273817110352045311572590042129422185637614246640536714981862725
966913701837243884900502824772738682735066921588171072863954432
key_at_alice end: 710943487911513630243895542864209967984449
key_at_bob end:  710943487911513630243895542864209967984449
```

some more sophie germain primes are shown below -:

```
{'iter': 127, 'sgp':
1256186186751392336457834117209730520840567025191628388630761449700551805207962231829172866143814680807
2833841992718854119840701991506063663984854959536491, '2p+1':
2512372373502784672915668234419461041681134050383256777261522899401103610415924463658345732287629361614
5667683985437708239681403983012127327969709919072983}
{'iter': 126, 'sgp':
1258747093480571656674050927544133653091361582128828525186570828523426590827409554391689194998666944679
5260997875933215479312967870674178391549464815852293, '2p+1':
2517494186961143313348101855088267306182723164257657050373141657046853181654819108783378389997333889359
0521995751866430958625935741348356783098929631704587, 'generator':
1232780130749184299276315175019676014460644260255128138184628914547832139562684713531191293813376303080
27033690365234641163590607071834919693371037138353334}
{'iter': 93, 'sgp':
1334456061388568301695607048224311235633451003432872035283531362162993068230170235455599216602853214359
00182052186120084223281072200678129063118868507983839, '2p+1':
2668912122777136603391214096448622471266902006865744070567062724325986136460340470911198433205706428718
0036410437224016844656214440135625812623773701596779, 'generator':
9296234311954085008202499532738171103520453115725900421294221856376142466405367149818627259669137018372
4388490050282477273868273506692158817107286395443328}
```

codes for question 1, 2, and 3

code-1

```python
from Crypto.Hash import SHA256
import random,time,os

def find_hash(text):
    new_hash = SHA256.new()
    new_hash.update(bytes(text,'utf-8'))
    return new_hash.hexdigest()

def condition(text1,text2,first_bits=4):
    return (text1[:first_bits] == text2[:first_bits])

def generate_random(num):
    text = list('abcdefghijklmnopqrstuvwxyz0123456789.')
    random_bits = ''.join(random.sample(text,num))
    return random_bits

def write_to_file(filename,iterations,array):
    with open(filename,'a') as f:
        string = f'iterations: {iterations} ----------\nlen: {len(array)}\n{"
".join(map(lambda x:str(x),array))}\n\n'
        f.write(string)
```

```python
def generate_pairs(size1,size2):
  p1 = generate_random(size1) + '@iitj.ac.in'
  p2 = generate_random(size2) + "@iitj.ac.in"
  return p1,p2


h1 = find_hash('sharma.59@iitj.ac.in')
print(h1)

def n_calls(h1):
   t1 = time.perf_counter();

   l = []
   iterations = 0
   max_iterations = int(1e6)
   while (True):
      generated_value = generate_random(9) + '@iitj.ac.in'
      h2 = find_hash(generated_value)
      iterations += 1
      if (condition(h1,h2)):
         l.append(generated_value)
         break;
      if (iterations > max_iterations):
         break;

   if (len(l) > 0):
      write_to_file('output1.txt',iterations,l)
   else:
      print('len = 0')

   t2 = time.perf_counter()

   print(f'completed in {(t2-t1)/60:.2f} Mins')
   return iterations;

def find_pairs(values_match):
   t1 = time.perf_counter();

   l = []
   iterations = 0
   max_iterations = int(1e6)
   values_map = {}
   while (True):
      generated_value1 = generate_random(9) + "@iitj.ac.in"
      h1 = find_hash(generated_value1)
      iterations += 1
      if (h1[:values_match] in values_map):
         l.append((values_map[h1[:values_match]],generated_value1))
         break;
      else:
         values_map[h1[:values_match]] = generated_value1;
      if (iterations > max_iterations):
         break;
```

```python
    # print(len(l))

    # print(l)
    if (len(l) > 0):
        write_to_file('output2.txt',iterations,l)
    else:
        print('len = 0')

    t2 = time.perf_counter()

    print(f'completed in {(t2-t1)/60:.2f} Mins')
    return iterations;

# check_iterations(h1)
filenames = ['output1.txt','output2.txt']
for filename in filenames:
    if (os.path.exists(filename)):
        os.remove(filename)

output1_iterations = 0
output2_iterations = 0
n = 10;
for i in range(n):
    output1_iterations += n_calls(h1) / n
    output2_iterations += find_pairs(4) / n

with open('output1.txt','a') as f:
    f.write(str(int(output1_iterations)))

with open('output2.txt','a') as f:
    f.write(str(int(output2_iterations)))

exit(0)
```

code-2

```python
import random
import sys
sys.setrecursionlimit(1030)

def extended_eucledian(a,b):
    if (a%b == 0): return (b,0,1)
    val = [[a,1,1,0],[b,a//b,0,1]]
    while val[-1][0] != 1:
        new_val = []
        new_val.append(val[-2][0] % val[-1][0])
        if (new_val[0] == 0): return (val[-1][0],val[-1][-2],val[-1][-1])
        new_val.append(val[-1][0] // new_val[0])
        new_val.append(val[-2][-2] - val[-1][1] * val[-1][-2])
        new_val.append(val[-2][-1] - val[-1][1] * val[-1][-1])
        val.append(new_val)
```

```python
        return (val[-2][1],val[-1][-2],val[-1][-1]);

class rsa_algorithm(object):
    def __init__(self):
        # self.p = 108888694504183521607680000001
        # self.q = 26525285981219105863630847999999
        self.p = self.generate_prime();
        self.q = self.generate_prime();
        # self.p = 155;
        # self.q = 21711;
        print('P: ',self.p)
        print('Q: ',self.q)
        self.phi_n = (self.p-1)*(self.q-1)
        self.n = self.p * self.q
    def find_e(self):
        for i in range(3,100,2):
            gcd,x,y = extended_eucledian(self.phi_n,i)
            if (gcd == 1):
                return (i,y)


    def rsa_encryption(self,msg):
        self.e,self.__d = self.find_e();
        # print('inside d:',self.__d)
        if (self.__d < 0):
            print('d < 0, making it positive !!!')
            self.__d = self.phi_n + self.__d
        encoded_string = []
        encoded_msg = bytes(msg,'utf-8')
        for char in encoded_msg:
            encoded_string.append(self.modulo_exponentiation(char,self.e,self.n))
        return ''.join(map(lambda x: bytearray(hex(x),'utf-8').decode() + ' '
,encoded_string))[:-1]

    def rsa_decryption(self,cipher):
        decoded_msg = []
        for i in cipher.split(' '):
            cipher_msg = i.strip();
            cipher_msg = int(cipher_msg.encode(),16)
            # print(f'cipher msg: {cipher_msg}')

decoded_msg.append(self.modulo_exponentiation(cipher_msg,self.__d,self.n))
        # print(decoded_msg)
        # return
        return ''.join(map(lambda x:chr(x),decoded_msg))


    def square_multiply(self,bas,exp):
        if (exp == 0): return 1;
        if (exp == 1): return bas;
        if (exp & 1):
            return bas * self.square_multiply(bas * bas, (exp-1)//2)
        else:
            return self.square_multiply(bas * bas,exp//2)
```

```python
    def modulo_exponentiation(self,bas,exp,N):
        if (exp == 0):
            return 1;
        if (exp == 1):
            return bas % N;
        t = self.modulo_exponentiation(bas, exp // 2,N);
        t = (t * t) % N;
        if (exp % 2 == 0):
            return t;
        else:
            return ((bas % N) * t) % N;

    def generate_random_512(self):
        return int("1"+ ''.join([random.choice(["0","1"]) for _ in range(510)])  +
"1",2)

    def find_s_t(self,num):
        new_num = num - 1;
        s = 0;
        while (new_num & 1 == 0):
            s += 1;
            new_num >>= 1;
        return (s,new_num)

    def miller_rabin_test(self,num,s,t):
        new_num = num - 1;
        aa = random.randint(2,new_num)
        b_i = self.modulo_exponentiation(aa,t,num);
        if (b_i == 1 or b_i == num - 1):
            return 1;
        pow_2 = 0;
        while (True):
            pow_2 += 1
            if (pow_2 == s+1):
                break;
            b_i = self.modulo_exponentiation(b_i,2,num);
            # if (b_i == 1): return 0;
            if (b_i == num - 1): return 1;
        return 0;

    def _check_prime(self,num):
        s,t = self.find_s_t(num);
        for i in range(100):
            vote = self.miller_rabin_test(num,s,t)
            if (vote == 0):
                return False;
        return True;

    def generate_prime(self):
        a = False;
        final_prime = None;
        # iterations = 0;
        while (not a):
```

```
            # iterations+=1
            final_prime = self.generate_random_512()
            a = self._check_prime(final_prime)
        # print(iterations)
        return final_prime;

    def fast_modulo_exponentiation(self,base,power,N):
        dp = [-1]




original_text = "Nirbhay Sharma is a good boy !!! and he studies at IIT Jodhpur
and he likes to do exercise and yoga";
rsa = rsa_algorithm();

# print(rsa.generate_prime())
# print(rsa.generate_prime())
# exit(0)

enc = rsa.rsa_encryption(original_text);

print('private d:',rsa._rsa_algorithm__d)
print('public n: ',rsa.n)
print('public e:',rsa.e)

print('#'*25 + ' original_text ' + '#'*25)
print(original_text)

print('#'*25 + ' encrypted text ' + '#'*25)
print(enc)

dec = rsa.rsa_decryption(enc)
print('#'*25 + ' decrypted text ' + '#'*25)
print(dec)
```

code-3

```
import random, json, os

def modulo_exponentiation(bas,exp,N):
    if (exp == 0):
        return 1;
    if (exp == 1):
        return bas % N;
    t = modulo_exponentiation(bas, exp // 2,N);
    t = (t * t) % N;
    if (exp % 2 == 0):
        return t;
```

```python
        else:
            return ((bas % N) * t) % N;

    def square_multiply(bas,exp):
        if (exp == 0): return 1;
        if (exp == 1): return bas;
        if (exp & 1):
            return bas * square_multiply(bas * bas, (exp-1)//2)
        else:
            return square_multiply(bas * bas,exp//2)

    def write_to_file(filename,dictionary):
        with open(filename, 'a') as f:
            f.write(str(dictionary) + "\n")

    def find_generator(prime_p):
        while True:
            random_a = random.randint(2,prime_p-1)
            rem = modulo_exponentiation(random_a,prime_p-1,prime_p)
            if (rem == 1): return random_a;


    class Prime_generator_machine(object):
        def __init__(self,bits_in_prime):
            self.bip = bits_in_prime;

        def modulo_exponentiation(self,bas,exp,N):
            if (exp == 0):
                return 1;
            if (exp == 1):
                return bas % N;
            t = self.modulo_exponentiation(bas, exp // 2,N);
            t = (t * t) % N;
            if (exp % 2 == 0):
                return t;
            else:
                return ((bas % N) * t) % N;

        def generate_random_bits(self):
            return int("1"+ ''.join([random.choice(["0","1"]) for _ in range(self.bip-
    2)])  + "1",2)

        def find_s_t(self,num):
            new_num = num - 1;
            s = 0;
            while (new_num & 1 == 0):
                s += 1;
                new_num >>= 1;
            return (s,new_num)

        def miller_rabin_test(self,num,s,t):
            new_num = num - 1;
            aa = random.randint(2,new_num)
            b_i = self.modulo_exponentiation(aa,t,num);
```

```python
            if (b_i == 1 or b_i == num - 1):
                return 1;
            pow_2 = 0;
            while (True):
                pow_2 += 1
                if (pow_2 == s+1):
                    break;
                b_i = self.modulo_exponentiation(b_i,2,num);
                # if (b_i == 1): return 0;
                if (b_i == num - 1): return 1;
            return 0;


    def _check_prime(self,num):
        s,t = self.find_s_t(num);
        for i in range(100):
            vote = self.miller_rabin_test(num,s,t)
            if (vote == 0):
                return False;
        return True;


    def generate_prime(self):
        a = False;
        final_prime = None;
        # iterations = 0;
        while (not a):
            # iterations+=1
            final_prime = self.generate_random_bits()
            a = self._check_prime(final_prime)
        # print(iterations)
        return final_prime;


pgm = Prime_generator_machine(512)

def generate_sophie_germain_prime(pgm,thresh):
    sgp = None;
    iterations = 0;
    p_dash = 0
    while True:
        p_dash = pgm.generate_prime()

        # print(f'p_dash prime_test: {pgm._check_prime(p_dash)}')
        iterations += 1
        print(f'iterations: {iterations}',end='\r')
        sophie_germain_prime = 2*p_dash + 1
        if (pgm._check_prime(sophie_germain_prime)):
            sgp = sophie_germain_prime
            break;
        if (iterations == thresh):
            break;
        # print(f'sophie germain prime test:
{pgm._check_prime(sophie_germain_prime)}')
```

```python
        # print('p_dash:',p_dash)
        # print('generated sophie germain prime:',sophie_germain_prime)
    # print('p_dash:',p_dash)
    return ((p_dash,sgp,iterations) if sgp is not None else
("None","None",iterations));



thresh = 1000
sophie_germain_prime,condition_sophie_germain,iterations =
generate_sophie_germain_prime(pgm,thresh)
if condition_sophie_germain == "None":
    print(f'cannot find sophie germain prime till {thresh} iterations !!! Try
running again')
    exit(0)
print('sphie_germain_prime: ',sophie_germain_prime)
generator = find_generator(sophie_germain_prime)
write_to_file('sophie-germain-prime.txt',
{"iter":iterations,"sgp":sophie_germain_prime,"2p+1":condition_sophie_germain,"gen
erator":generator})
# for a prime order group every element is a generator
print('found generator: ',generator)

# exit(0)

class Alice(object):
    def __init__(self,g):
        self.a = random.randint(3,10)
        self.g = g

    def _generate_key(self):
        return square_multiply(self.g,self.a)

    def _generate_final_key(self,bob_g_random):
        return square_multiply(bob_g_random,self.a)

class Bob(object):
    def __init__(self,g):
        self.b = random.randint(3,10)
        self.g = g

    def _generate_key(self):
        return square_multiply(self.g,self.b)

    def _generate_final_key(self,alice_g_random):
        return square_multiply(alice_g_random,self.b)

def diffie_hilmann_exchange(g):
    alice = Alice(g)
    bob = Bob(g)

    alice_generated_key = alice._generate_key()
    bob_generated_key = bob._generate_key()
```

```python
        final_key_at_alice = alice._generate_final_key(bob_generated_key)
        final_key_at_bob = bob._generate_final_key(alice_generated_key)

        print("key_at_alice end:",final_key_at_alice)
        print('key_at_bob end:',final_key_at_bob)
        # print(final_key_at_alice==final_key_at_bob)


diffie_hilmann_exchange(23)
```