Homework #4

Due date: 26 November 2017

1. (**20 pts**) The composite number n is the product of two large primes, namely p and q. Assume that nobody knows the factorization of n. Consider the following function

   $H(x) = x^2 \pmod{n}$.

   H(x) is one-way function since it is difficult to compute the square root of x modulo a composite number when the factorization is not known.

   a. Explain why H(x) is <u>not</u> a good cryptographic hash function by discussing the properties of cryptographic hash functions. (**Hint:** Discuss non-invertibility, weak and strong collision resistance.) (**10 pts**)

   b. Explain why H(x) is <u>not</u> even a good hash function by demonstrating its output is biased. Explain your answer using on the example, where n = 15, i.e. p = 3 and q = 5. (**10 pts**)

2. **(30 pts)** In order to increase security, Bob chooses the modulus n and two encryption exponents, e_1 and e_2. He asks Alice to perform double encryption, i.e. to encrypt her message m to him by first computing $c_1 \equiv m^{e_1} \pmod{N}$, then encrypting c_1 to get $c_2 \equiv c_1^{e_2}$ (mod N). Alice then sends c_2 to Bob.

   a. The double encryption does not increase security over single encryption and in fact is equivalent to single encryption with a private key e_3. Explain why? (**10 pts**)

   b. Consider the following
   p =
   5101992342203516357697535790036406465112696833686666892830644684923604789578858837330734668955362945351024616140475938505160037019389600812468312274731287
   q =
   10467785804023663154081159890916670751197684284050596453663074112949922468570225369830419350044296830552368635876390157350061674041677409479215017880761471
   e_1 = 65537
   e_2 = 65539
   c=
   49346329369462854363271782448315211099681514434640599974621854671392427139371635210345363031473153599464348320842322740635553960642953510895196069229258786536891749021841437200659026184844350444165516427177404248892439049448334399411189045056185102422704738091759473313311163625490140702356058569205275317138

where c is the ciphertext as a result of the double RSA encryption with e_1 and e_2. Find the equivalent decryption key d_3 and decrypt the ciphertext with a single modular exponentiation operation. (**20 pts**)

3. (**20 pts**) Consider the following RSA parameters:

N:
59300007034639909939573758056469081496649095362931218335710263559636829491463214675399654309454248418861314244988626244340386564274798699371224710226178573100179350623455240677742280755859388396827300307411521308725481313615050599976223074746012316066224478374065904742491869819255200529839123356150617924773
e: 65537

Consider also the following ciphertext,

c:
47446751200838582684511548326026682588511648894181169756283983741637372571242927061754188680616949591184277685045578068075300868535431681867533253595068603738094583234430020018598479411043102722508296983050330949598947760334702343661636595593088242435796182825602462354582903741935054242424335737738087473281,

which is the encryption of my PIN of four digits. The textbook RSA without padding is used. Find my PIN.

4. (**30 pts**) Consider the following security game. Suppose that an attacker wants to decrypt the ciphertext c encrypted using the RSA algorithm and obtain the plaintext m, where $c = m^e \bmod N$. She knows neither the private key d nor the factorization of the modulus N. However, she can query an oracle (e.g., a program running on a remote server) with a ciphertext $c' \neq c$, and receives the corresponding plaintext $m' = c'^d \bmod N$.

a. The attacker can decrypt c and recover m. Show how. (**10 pts**)

b. Consider the following RSA parameters and a challenge ciphertext

N:
140551657748123311843904632833471669259677424177527429472076641459146867906661942068298379563181123587514898171198345842702518087577996533400370831952801883330035838563895672878817032429070070491801818686348972449259014970161691017147789125584023630380720107022841065881140401317726964037566281222748970712203

e: 65537

c:
107370145208181012882157035957831285007639861193502148958526088911145

1113123548578796513152984304917060579273635847036996508323650831497303880588177162273515806437805969230320212726501977058000133014358403790665132037058833111881196671090834779351294250383674721013894361656885598379010787464300281393535909 88

The instructor (accessible via erkays@sabanciuniv.edu) is the oracle that you can submit your queries $c' \neq c$ via e-mail. Your queries must be in the following form

```
c' = int("your query here as an integer")
```

I challenge you to find *m*. (**20 pts**)

Notes:

i. You can use the Python function **pow**(m, e, N) to compute modular exponentiation.
ii. You can use the following two Python functions to compute gcd and modular inverse

```
def egcd(a, b):
    x,y, u,v = 0,1, 1,0
    while a != 0:
        q, r = b//a, b%a
        m, n = x-u*q, y-v*q
        b,a, x,y, u,v = a,r, u,v, m,n
    gcd = b
    return gcd, x, y

def modinv(a, m):
    gcd, x, y = egcd(a, m)
    if gcd != 1:
        return None  # modular inverse does not exist
    else:
        return x % m
```