Homework #4

Due date: 25 November 2015

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 not a good cryptographic hash function by discussing the properties of cryptographic hash functions. (**Hint:** Discuss non-invertibility, weak and strong collision resistance.) (**10 pts**)

It is neither weak nor strong collision resistant.

For instance, given x_1 and $h = h(x_1)$ it is easy to find x_2 with the same h since $x_2 \equiv -x_1 \bmod n$.

   b. Explain why H(x) is not 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**)

Not all numbers are quadratic residues. Therefore, the hash function will be biased, i.e. it will not produce non-quadratic residues.

In case of n = 15, compute the square roots of all numbers Z_{15}.

$1^2 \bmod 15 = 1$	$2^2 \bmod 15 = 4$	$3^2 \bmod 15 = 9$
$4^2 \bmod 15 = 1$	$5^2 \bmod 15 = 10$	$6^2 \bmod 15 = 6$
$7^2 \bmod 15 = 4$	$8^2 \bmod 15 = 4$	$9^2 \bmod 15 = 6$
$10^2 \bmod 15 = 10$	$11^2 \bmod 15 = 1$	$12^2 \bmod 15 = 9$
$13^2 \bmod 15 = 4$	$14^2 \bmod 15 = 1$	

As one can observe, the quadratic residues modulo 15 is 1, 4, 6, 9, and 10. This will generate a bias.

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 $c_2 \equiv c_1^{e_2} \pmod N$. 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**)

The double encryption is equivalent to single encryption with a private key e_3 such that

$e_3 \equiv e_1 \times e_2 \bmod \Phi(N)$ since

$c_2 \equiv c_1^{e_2} \pmod N \equiv (m^{e_1})^{e_2} \pmod N \equiv m^{e_1 \times e_2 \bmod \Phi(n)} \pmod N \equiv m^{e_3} \pmod N$.

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**)

e3 = 4295229443

d3 =
43545076186247478819850395715321242637491396910097868244257041430987271283284469058310751057703074287901664879382491646446144974525889364800512625958992416627826485902361497629831229361772526477754750069896212273449443327395455699512545376844442291251878100942455124024977411248574149643838270125336658474667

Decrypted message:
19630312218990746556545309514251776975922663273447818615403962868409498201061740895170619300387064408280067944012061487819737578225990517793004567110739990372933833182594778205299695759534005206540525678359293574243732415850845763772497008751145859601874700230380247264380230950144927086684834148394899215621

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

N:
593000070346399099395737580564690814966490953629312183357102635596368294914632146753996543094542484188613142449886262443403865642747986993712247102261785731001793506234552406777422807558593883968273003074115213087

2548131361505059997622307474601231606622447837406590474249186981925520
0529839123356150617924773
e: 65537

Consider also the following ciphertext,

c:
1024957790133807760214937258921900902752054740603696595549234323352777
8501417117186234248227645276237662944701269768063176721930294279490845
9039391947479890785966678888076552650597762354800086721987610097777666
4940026669802188775810027255406601226291695952742139543890850972212266
9135093419597700044377061460,

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

PIN Found: 1524

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**)

There are different ways. For example, the attacker picks a random k with $\gcd(k,N)=1$. She first encrypts it $\kappa = k^e \bmod N$. She performs the following operation $c' = c\kappa \bmod N$ and queries the oracle with c' and receives $m' = c'^d \bmod N$. Since we have

$$
\begin{aligned}
m' &= c'^d \bmod N \\
&= (c\kappa)^d \bmod N \\
&= c^d \kappa^d \bmod N \\
&= c^d (k^e \bmod N)^d \bmod N \\
&= c^d k^{ed} \bmod N \\
&= mk \bmod N
\end{aligned}
$$

She then computes $k^{-1} \bmod N$ and performs the following

$m'k^{-1} \bmod N = mk\,k^{-1} \bmod N = m$.

b. Consider the following RSA parameters and a challenge ciphertext

N:
1405516577481233118439046328334716692596774241775274294720766414591468
679066619420682983795631811235875148981711983458427025180875779965 33

400370831952801883330035838563895672878817032429070070491801818686348
972449259014970161691017147789125584023630380720107022841065881140401
31772696403756628122274897071 2203

e: 65537

c:
107370145208181012882157035957831285007639861193502148958526088911145
111312354857879651315298430491706057927363584703699650832365083149730
388058817716227351580643780596923032021272650197705800013301435840379
066513203705883311188119667109083477935129425038367472101389436165688
559837901078746430028139353590988

The instructor (accessible via erkays@sabanciuniv.edu) is the oracle that you can send your queries c’ ≠ 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**)

Using the method in (**a**)

k:
787566151485193271810211641729727382405637271412709035282818393616553676 04
054550155267160799586698196100972375786516685577069105479105909874509756 30
197355738973636211596730907251948195058730111424894170891743098399613607 91
235655195620513855043667094553106618794700078510538949729291975288181644 04
309387851047

k^{-1} mod N:
191234599102853733688665994251706563568818111023268239433849790691440137 46
567296475517531549274368506532240858710695034596238514500143646559842372 54
365549136674624794792373907560750811897783295895364462724511997963167731 63
452929158029745054906380834450704163874244531851459113116893666032780925 46
241473593566

k^e mod N:
130901251129824484919130423331250129585083236162647865366231483796772074 19
247895046748524527524214017052365094552465677980700798592323700608852503 86
681252740622993882140821223720202913628086119162646305814088208833789113 86
890950192431466098534696080146534766087442910326424241365359081237997982 47
2412956435535

c' = ck^e mod N:
879875008416568347506441443245302507780570712441156690933599692418974881 77
344830901731049779451961433836353012062666119003383399168697645167362571 45
451518430498614640677050921653774731800132299177392299282982928881333958 48

06976336875219892573886759349953281196904854905360080954753889344909276655
107541635458

$m' = c'^{d}$ mod N:
13036885816521250387151092210610699941068227679771253662726414915396148875
96962965749006782109581909961420196980951352776268292319613293184363252130
28964146936964948206475488032147958944189249342021898387876634312355913445
18196288122246689102474479114268663360945998813519279629196097351634440308
5770750507562

$m = m' k^{-1}$ mod N:
75230524638317440049788719860328164905560972661567442280286739083022716797
40115365430299679545820622527654232994139872159117995532380124020235849063
64909098706135018720675451285279288302337483701053411019877727750693097246
29565702113913754214061974147087933676493917191737304737323965900939830794
25228189179

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
```