Homework #4

Due date: **10 December 2021**

Notes:

- Note that there are five attached files: "RSA_Oracle_client.py" for Question 1, "RSA_OAEP.py" and "RSA_OAEP_client.py" for Question 2, "ElGamal.py" for Questions 3 & 4 and "DSA.py" for Questions 5 and 6.
- You are expected to submit your answer document as well as a separate Python code for each question. Do not modify the source codes that are given to you and do not submit them. You must import them to your sources as they are, do not solve the questions in those files.
    - Source files to submit: Q1.py, Q2.py, Q3.py, Q4.py, Q5.py, Q6.py
- Print out your numerical results in integer format, without "-e". (We do not want to see results like 1.2312312341324523e+24).
- Winzip your programs and add a readme.txt document (if necessary) to explain the programs and how to use them.
- Name your winzip file as "cs411_507_hw04_yourname.zip"
- Create a PDF document explaining your solutions briefly (a couple of sentences/equations for each question). Also include your numerical answers (numbers that you are expected to find). Explanations must match source files. Please also add the same explanations as comments and explanatory output.

1. (**20 pts**) Consider a deterministic RSA Oracle that is implemented at the server "cryptlygos.pythonanywhere.com/RSA_Oracle/". Connect the server using *RSA_Oracle_Get()* function, and it will send a ciphertext $\boldsymbol{c}$ **,** modulus $\boldsymbol{N}$ and public key **e.** You are expected find out the corresponding plaintext $\boldsymbol{m}$. You can query the RSA Oracle with any ciphertext $\bar{\boldsymbol{c}} \neq \boldsymbol{c}$ using the python function *RSA_Oracle_Query()*, and it will send the corresponding plaintext $\bar{\boldsymbol{m}}$. You can send as many queries as you want as long as $\bar{\boldsymbol{c}} \neq \boldsymbol{c}$. Then, check your answer using *RSA_Oracle_Checker()*
You can use the Python code RSA_Oracle_client.py to communicate with the server.

I choose randomly r = 30. It does not matter what did I choose. We know m1 and m2 are multiples of each other. So encrypted c1 and c2. Since m2 = (m**e * x**e) mod n and we e*d = 1 mod phi(n) .when we take e*d as exponent (r**ed *m **ed) mod N = r*m mod N. So, m_ = r*m mod N. As a result I used relationship between m1 and m2 ,encrypted c1 and c2 applied formula and change base of formula as rm
e*d = 1 mod phi(n) change base as m*r
(r**ed *m **ed) mod N = r*m mod N = m2
{'m_':
18287477167346339561913746475360599955421600698845450651759703375514552203814128795969387063288218065129092150}
message:
609582572244877985397124882512019998514053356628181688391990112517151740127137626532312902109607268837636405

b'Bravo! You find it. Your secret code is 65695'
messagetext: Bravo! You find it. Your secret code is 65695
Congrats

2. (**20** **pts**) Consider the RSA OAEP implemented at the server http://cryptlygos.pythonanywhere.com/RSA_OAEP. By using the *RSA_OAEP_Get()* funtion get your ciphertext (**c**), public key (**e**) and modulus (**N**). The RSA-OAEP implementation at the server is given in the file "RSA_OAEP.py", in which the random number R is an 8-bit unsigned integer.

   I select a random four decimal digit PIN and encrypt it using RSA. Your mission is to find the randomly chosen PIN. You can use the Python code RSA_OAEP_client.py to communicate with the server.

   since c, N and e are given in the question and I know range of m (4 digit so range is 1000) and R I range (2*7,2*8-1) since 8 bit unsigned integer. also given the source code. By using nested loop I found m and R. I check my answer to RSA_OAEP_Checker. It is correct.
   PIN_ : 8211, R : 130

```
{'c':
11352871632598964629356838860088839410463425369482314036274872077619009
180504, 'N':
75912732707060243642078909648401302780483043992228012220203806825283170
905549, 'e': 65537}
M =  8211
R =  130
Congrats
```

3. (**10 pts**) Consider the ElGamal encryption algorithm implemented in the file "ElGamal.py", which contains a flaw. We used this implementation to encrypt a message using the following parameters:

   q = 21951366550493000718261853105201996539940614036945856492989212434043
   p =
   19088517355831130409934748378779805905302705641965364159942125090101617
   99975302701818966798354528918628523029441864160503682524067809739739523
   04043595070715594043892482165997787657197138464583053698006362742220168
   30546342326148389081626885189560815658702360521026851058633029338654212
   43714153248167871823317078513925980737349818011430897694478440665517540
   06255395274462668748032735120045083082893838173662613857114711446785176
   08879306099468067802238512104280975543754563520312324843005014163082784
   86706749306383133223230788507203013300913700383314510553853774057307370
   0482783653386269696771584777005458361676730195269
   g =
   11087495091603530820155897131840473261858409884534929785120936970350376
   59834044289688357464174469647962847262434235927215163753913689593361793
   90305203914071844167089070134876151918097100789097413593600008572563950

4489286304481512538471634743581306619581548562429171944346668887904942
4007911844446338806244185922330210312707943825392115584470267705273405
1802239534092327598870055016955978239282077553105785546036019197246668
0952781636988810881151087847853847227509215418323947988173634260322565
0419112774665516766240118002532725372593205923027699351646411194158590
76391606505106857606401173246121593239944724883731441O

public key (h) =
9873203779178244917358573822108714866225564575093808381213522055099626
9050485747183770798439918395103889424471338469676812894568199473706838
2628603050350281850071815481158793767153887039702039469691684647739659
8261743047774222401970648745921252876334052024434425956082895113727370
0274728900091849089034842727497404819151992046576439037554513575801211
4725748919054090278301411984618992540495713579651863596237507088245986
5458127443035772212293255428264034049635490612261573387887524888802067
1649049934849373218564734793637576605482517554325754533107701049732871
9450203901452547531512637567611587435903881933071324129 1

And the resulting ciphertext is

r =
1323916546229624747319828608497338386499637825313655774042952171914406
3831821729177799399305950848429010164591542578562684023946241096301139
7468731251853378852497076591684939427893138197446673356190457539005062
4036913608662385950023601834508728625976944718994802038057143373745491
9151202571220238401902685316173160652664728814802730286862716501856130
4748615978345573944100642206602629510864733812138720010018120892451772
9532399908762220109812468390106470023643489876321036806656562004998490
3773379389046508917805706757074599173912975537261341906213763409479276
421831618970202312256091485549555360990937615951232838268

t=
2928045218382913457443684144012119252158702606935650406082722909043750
2220635359525684453992465493386865359058375133613624323523104279156170
0988951279870088942537729008874317926727314888524372408731094089634092
1032631826293301818196131007722811813618752139364912766264315045693874
4441329575160845769037179912151259217346950270911675287724367661348630
4301791824830771168126440524765508203739226544881266746681692174908432
8029750765491112978016172077944711575384560448985809167799882976912165
8248700257907284597750086859650710791436439445885497573393366992798616
53817076680172669373971736606326598544835667466271193204

Can you find the message?

Yes, since we know k is a random number between 1 and q-1. I create a loop to find k depend on r = g^k mod p equation. When I found k, I used it and

t = h^km mod p equation to find message. My k and message are:

k : 64278

message: I am gonna make him an offer he cannot refuse

4. (**10 pts**) We encrypted two messages m_1 and m_2 using ElGamal encryption algorithm given in “ElGamal.py”, however, it contains a flaw and we lost m_2.

q = 1367176787662174613885987459588219879372220953507
p =
1236733238912428671775140968195802364654972740634589843252410558440111532241450548588645213717746491050188229711169826507159411178810355528157263378277436105673943193226513548278903934103462870948679494713306496113531443502802028039946992528511896493717366562796189559886509040924552324059720312380330175551 79

g =
4046212211887218194537892135248707293951695201866273568605548461473137730024129679327419401533067787456191386225703234931919531045428857993189778317269936847661764393559206653659296420399676461739349776303010245464555388958985850815586365900824413845073410303433065370488641520364521819850209141074874035164 5

$(message_1, r_1, t_1)$ = (b'I am gonna make him an offer he cannot refuse',
3581366112733152746935840006160236246858488405594752354230362272157249908666312496219775973257012829953850416555947179942023015615136135565768776884255820791300017307008154557978275675911618093249844839025940263200250690404275536972391325034631939746192469516544708632741660065987666998236474954030084361297 0,
4362722421811579722828924947592103290703422046049235611189550393608285441816825041649707294134747199474372797325464232558545358408174079990636029984404180265422032843141195161667934699555984326425462806850161329295945458211855680566634769173371492203158961416510737988377559694479865363170581179508924177112 3)

$(message_2, r_2, t_2)$ = (b'????????????????????????????????????',
3581366112733152746935840006160236246858488405594752354230362272157249908666312496219775973257012829953850416555947179942023015615136135565768776884255820791300017307008154557978275675911618093249844839025940263200250690404275536972391325034631939746192469516544708632741660065987666998236474954030084361297 0,
5956813619278201140800924272068090554564909295979962450630656519740430477504942407392993409599925085203647233193851018647091657112428687538063606647173251762864208428373009582042639482695547913744731417387816335621025572344854442283807515125579921298217666328301529878649319957708915200631529408344007611288)

Can you recover m_2 using the given settings? If yes, demonstrate your work.

Since r1 = r2 = r and m1 is known plaintext, we can find m2 by using the formula: $t_1/m_1 \equiv \beta^k \equiv t_2/m_2 \pmod p \Rightarrow m_2 = (t_2 m_1)/t_1 \pmod p$. you can see my code implementation of Q4.py. my answer is:

```
m2:
9692830212984936551784619824161327940136710295426995348267946161081657 8
6147584985094094249886701190566804838203669396484740329991713
Well, it was more like a command, no was not an option!
```

5. (**20 pts**) Consider the DSA scheme implemented in the file "DSA.py". The public parameters and public key are:

   q = 2105046191516306400569847275281846796048466422241946124042290558 7329
   p =
   166350012684247702483624960208789827948559730427271231102762182213631 15
   248532121656926998495324422884201900546908247299422529867020257795305 91
   560339911978412381797755147593874984363074709080555130318782311891787 01
   070612920135277680943020983237055039783199518335674190378326362190994 72
   142540390679316084341253020797072674386750831658161591413643898670788 05
   040919735263251876251210482740557647013129583231819964879586171515478 60
   175051687922070906846393742742561234022198080875417223026819462689334 84
   824286411774800000044873249941828965744594817357937388103636107551213 52
   0018336143107199947863581191230243194303971728193
   g =
   409316209796607490968734702020982808779338767387886312652237001531251 35
   193841427809041833508791364995779974869305845643613737878921303102160 18
   419844953120490173178167961210626495302179022102966346401633568701341 81
   395034601787974591435423590515009773320665843668567792787586385542473 70
   781338560822880124448745156416572503889588206898909656464583234191399 46
   043329041958728315521559242956876862378305965351919813096282501399946 65
   077358186000006346169433722976057429006410973538262567497727108655658 41
   154850053915645064501118147270775998865117868917286567042196426174136 39
   0644409217272929476468594666431679103077284382 76
   public key - beta =
   333000142450304432593219767613612726668939199509191301567259738108836 31
   954436364362141005905089953196434884271254246865239480441372592023389 07
   646703934598548017408323184017924886648962864744316964924113022186198 95
   035696390789148524777716364767139044719785911738347323622486681245728 14
   187471096595291100828182776312107520878628412160757411678899054880896 69
   423185704020852396281310433750953573745705629323883712097669867644208 37
   217747742747867891059071710943307144176311820220665601335449187640557 81
   833401304792723941664262856342897852021019553761633829580915470090064 19
   0807162250366281473086392726699036793621049176 51

   You are given two signatures for two different messages as follows:

(message₁, r_1, s_1) = (b"Asking questions during the lectures helps you understand Crypto", 260444855760506318805841590364189311211267498403457607938240440795, 15045429964567421250403275656320025283600046882519690784113588548158)
(message₂, r_2, s_2) = (b"Keep your friends close, but your enemies closer", 260444855760506318805841590364189311211267498403457607938240440795, 14016151436550334193141059702675072658308100333231844563375725796770)

Can you find the private key?

Since r1=r2=r. I used $a = (s_1h_2 - s_2h_1)(r(s_2 - s_1))^{-1} \bmod q$ to find private key(a). to find h1 and h2 I used SHAKE128. R and s1 and s2 are given in the question. So I just apply equation. My answer is

```
private key:
18011493590957919843196654272530256451916130571913898417508651137437
```

6. (**20 pts**) Consider the DSA scheme implemented in the file "DSA.py". The public parameters and public key are:

q = 1274928665248456750459255476142268320222010991943

p =
94399082877738640356344835093633851742226810946548058167594106609599304101483376198601628644645578978665867743371516213549559017509270013785262825124888169738692088560919995075509146379802866347021353299579995280712578946802331952341703103059527013530389111994085951544456654086033481582042901134498773988127

g =
74757613048887093209741634228228425902948572222965683892966782829654298800791789084861356704346371244921201938818880899647348974925451953450279300514594642896343751389085838466583384452902564477981127117505259585303938871436241327714244689153971542398500058515599232922200606171788427214873986464441516423273

public key - beta =
9391078822012222264248483853957955450074521847096866533459681369546944886235023857738438187102424298184377435789154539420500484576343932422250732759800837979336463896251863203597988162906413924736488554239908614170057127399588501615428907239954984946982024571938034476806841633488050802767414373595444261997

You are given two signatures for two different message as follows:

(message₁, r_1, s_1) = (message1 = b'Erkay hoca wish that you did learn a lot in the Cryptography course', 780456265196245442017019073827244628033034896446, 214154189471546244965139202160125045302874348377) (message$_2$, r_2, s_2) = (b'Who will win the 2021 F1 championship, Max or Lewis?', 927294142715241205623350780659879368622965215767, 151110642214296558517943730901561426792280910589)

Can you find my private key? (**Hint**: I ran out of random numbers for the signature of the second message)

Yes, since r1 != r2. We can claim that k2 = x*k1. I applied that formula:
a = (s1h2 – s2h1x)(s2r1x – s1r1)**-1 mod q. Since x in unknown which gives us the coefficient between k1 and k2. Firstly, I try loop range 1<x<50 and I could not find any answer so I increased the range 1<x<100. To check every possible of a. I used the formula: beta = g**a mod p.
So, my answer is:
x: 63
a: 66568624500090235129890566130399211243633217014

a is my private key and x is coefficient between k1 and k2: k2 = 63*k1