Homework #4

Due date: 18 December 2020

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 Question 3 and "DSA.py" for Question 4 and the bonus question.
- 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, Qbonus.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. (**30 pts**) Consider a deterministic RSA Oracle that is implemented at the server "cryptlygos.pythonanywhere.com/RSA_Oracle/<*your_id*>". When connected to the server, you will be sent a ciphertext $\boldsymbol{c}$ and expected find out the corresponding plaintext $\boldsymbol{m}$. You can query the RSA Oracle with any ciphertext $\bar{\boldsymbol{c}} \neq \boldsymbol{c}$, and you will be given the corresponding plaintext $\bar{\boldsymbol{m}}$. You can send as many queries as you want as long as $\bar{\boldsymbol{c}} \neq \boldsymbol{c}$. You can use the Python code RSA_Oracle_client.py to communicate with the server.

We find a random number x which is co-prime to N. We then calculate cprime by the following equation cpirme =(c * pow(x,e,N)) % N. We send this oracle server and get m_.
In order to find m we use the following equation m = (m_ * modinv(x,N)) % N)

The message is: b'Bravo: you find it. Your secret code is 80547'
M is:
6095825722450865819551734865562260598894378537706296808584382917210893821814324138861712673149938696017152 55

The code is available at Q1.py

2. (**30 pts**) Consider the RSA OAEP implemented at the server http://cryptlygos.pythonanywhere.com/RSA_OAEP/ <your_id>. The server will send you a 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. You need to connect to the RSA encryption server that challenges you with $(\boldsymbol{N}, \boldsymbol{e}, \boldsymbol{c})$. Your mission is to find the randomly chosen PIN. You can use the Python code RSA_OAPE_client.py to communicate with the server.

Since we know that the pin is 4 digits, we can use brute force attack on every 4-digit pin and find the same cipher text.
The pin is verified through the RSA_OAPE_client.py file.
Here are the results for R and PIN using brute force:
The PIN is: 3877 and R is: 218
The code is available at Q2.py.

3. (**20 pts**) Consider the ElGamal encryption algorithm implemented in the file "ElGamal.py", which contains a flaw. I used this implementation to encrypt a message using the following parameters:

q = 20229678282835322453606583744403220194075962461239010550087309021811
p =
1251504390939480375345041102264985427372153725101110774882681116845968062835139115448704132059500673623933219249223694396652305374447612772879796380815114250659533012062166337151828118120479783170734943655844313935567234782526772887937628967751726860995967123505922499478546360833066949445716325037358138003624765203096948104677201379927126871010448702216486500480286407606697415301212555106090605411292046986904522332957701593582486442861244672394204046530018591792330504203330631980971261887206379690413278828551849799932748592973092120274593593691383457761025429880920557516200502517087820078659075185000685792 1419
g =
2256483143741433163413007675067934542893022968337437312283381964942344365449719628255630752397325376452002398784394008507857025386943645437696558240874471345442532398588406749907930002481624160959132193798842426822193910104962138845873425590946341754334144292886002962901550160578482452138075339294826241799645761655320983735381974177635207208471824667516956679913974643342159550037320378814445802296879470561504511689460916200417902612323039671250567503846175990654512915878143201233050978046269551126178155060158781645062181955781969136435905570787457855530003987887049118699525033120811790739590564684316550493132
public key (h) =
1265126138933377994348793193477342224736956600354964713945582205290651827674605003741290400826146165752452701594226002213036650208863340321329798489264160728930653315907521926136642928347549825144026262035747350182493795559385070130959552499813885202334575993642935128132458545523498489490586883187848396314164874056757696154989511633927620869557222556876855999079308839417416012746206040455611002092520255736121673298963050693639916367968280807028975614596114022230524360150581344884219834519025619777858430431159461562871537004523472161672182851052258466610762884570310894027628303901161674783788320479747219000276

And the resulting ciphertext is
r =
3813677439444837990381281624769265484071989883494833765363155214071727573627590213038823018054653614040833306533736593789523636716088751609591517852868217052905415751457961942309213803782661174042131067555996860094296315483087375444362454092891960492098796234624392186112659124915872546640723139762874453050592110272036917039293020539724872406856066252779419482651672320132092421939867392668795959155312634804888215300607725584330531720210355201550529764936881761210810883102986464111409096572364185502722477587178710137175828696000683028806920671859797982157383943866111320227830105178421690303627627943337128795446
t=
1019203324011337764086016919505431598172751432732900879044413072910705693004729954775515077563623725236797903281542668544832920731815380702608045490828150101074425081803465167058347795273524849951218234416389270680829505886140615697680581705223591338576400818998304994727053010393703515214202183664553391314113575511407644381267719471957820539450086177571527442170169240201234909584949128688392025729006229726875115454037910877888186669741970100607416541816360185672659099598224418809136881121405838535630396747539327405509778193729694087461902772451105039708030362142700540320073669609676401363729100673775379411981 4

Can you find my message?

The session key k is small thus there is a flaw in encryption function. They key is 16-bit only therefore an exhaustive search or brute force attack for values in 2**16 will give us k. Then we can find the message using the equation m = (inv_h * t) % p where
inv_h = modinv(h**k,p)

Value of k is: 31659
The message is:
457793356560769121468996365489164659590358184640690762607708646930482042043545002053435644972301335121440011965170014542294903350314432332909128 8273353023
The decrypted message is:
b'Why is Monday so far from Friday, and Friday so close to Monday?'

The code is available at Q3.py.

4. (**20 pts**) Consider the DSA scheme implemented in the file “DSA.py”. The public parameters and public key are:

q = 184628707979587343584605403158023119637449999545068079815084986350 91

p =
2184410211212223748405848499022322252781698170282827917149814303658271627148547402838054269686219372085227261839750365877112811456843003454431183684813255659132427311783911547834305153842743766472298083077116193913922296470769527695743296803336535230208036631541573553211130271085780728179824904332089902780013512287312324374352472460207045796765728588456385896818773268072336990622221420125028844382472226168282897015873158766358517403288776798821914399671738092399809679406006402326458494911535471521137516886054471684394025988716816326250541344063298095236665669193523253872172645003708726385493517979869499934551 7

g =
13843079639351340920273184714590884400432847093058770970775133079628015343474638985949514224469231316509301786191837239734743524804707156837615319355419215945094865320399756037490734275197507243978890158231379210099367755690209217652326933425758170008835084657241675545571324146202714002127571892258435472678396358353938476569410849475658691697420643000086724156167275855286708191941521213998074404126295230559090196852525498568126029906179168789585152438330622252753643553805877257623433974639379577436808678860489830511416186993204671106346196262903362008285485594747047950971109814842643611103016670841253194356243

public key - beta =
6187481213658176498787124123601684091780046690985227386674127034254039365850646655310542241724937514112519192485497669738105144173607992347626869972509174309127140941080651743898030456747633487761927322752193676176314211884662768871783260572354989592156755352437101758031330846064492530779348477298394716501400849788380847680039744807953192006233069850428367974025006391433578254859633968702925514987402010031888483663325943692618870576893826021018783543580318493456251127341437691102522482919743872855098214539426447960934626890138798345418250945885432084267499991534185991486840567366979305573275554091497155603826

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

(message$_1$, r_1, s_1) = (b"He who laugh last didn't get the joke", 61645729931482682785443152461587949660612434566030814273897926987 84, 2412874836775368230194957659405258449579579568340501217618177629780)
(message$_2$, r_2, s_2) = (b"Ask me no questions, and I'll tell you no lies", 6164572993148268278544315246158794966061243456603081427389792698784, 3433793651282707205395973670954853011289701782741048461895987951 61)

Can you find my private key?

1- Signature verification fails with the present values Sig_Ver(msg1, r, s1, key, q, p, g)
2- Since r1 and r2 are same we can calculate the secret key as follows:
   $a = (s_i h_j - s_j h_i)(r(s_j - s_i))^{-1} \bmod q$
   Since modinv(r*(s2-s1),q) doesn't exist we can't find the secret key. The code is available at Q4.py.

Bonus Question

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

q = 1514133908421153778079840282146866825323385529325028247070748652 3729

p =
1545935267817019499905979795383594370376929979852264048594925102123006 1
2398729332865962816718750364447667672608251611563391423749531442646671 7
5663093532210016977000296281428180052962512096930034626707240943073909 4
2994856864717548964192394705552369066239727549981401165961593331300122 0
7335581801649930864723793258872094184390760368305959689481224635425654 8
8458285559269152814846930461678806155717771594791617514000333739836058 3
6719170230181709587371581076895039257660134543465104228249625889879829 3
8979163413156937317635345138712958701172946723054479401323331428941627 9
07591967042409728994120165930062230878713574049 69

g =
3800569625008648766049545537807478639158256666453837543156865205157342 4
5317519533829391451831838993251241919702249219326707246675459462046153 4
5673624978417100025991119530913449303439945034310716924005253545285479 1
8075410538790275781900267312641988973075426468087022427855954288858299 4
5892780888951898431749014172940178634272504225094118257474033479390191 2
9741702226040151773233688142649898356794070762899748555524143987796255 2
1837257916022552980027627057473062644879659632681204107806120144998907 9
9133891326633432116032465148401275244163414024346573093961924251528071 4
35687369996598536340201068685144339620001880019 9

public key - beta =
1381171819491288773125997368753165901722123307269375833932067755608596 1
0917415125343129913199909880123208951252731387994849304246563286189863 3
8233650799555131896857586001490595604365368085682743275712428137943225 1
1971562840589235730602915057458411978583232560567483880115464189574531 1
1612718894365028998464581319009883877772546761576721995259383264702443 6
3881227814557082187788046660952433631553517068095734365024876910709029 4
1685011485406404333887994054290193662496930324859520810879575122538720 3
4053957399410425706981647199730372613947643303141205096073444084858201 3
33073888826999550103201833184470656754878613221 41

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

(message$_1$, r_1, s_1) = (b"He who laugh last didn't get the joke",
780720772592321367005945670607735754560466840092435474685060772631 0,
1013741352181898186055829584414246324873628066967137660793977442016 9)
(message$_2$, r_2, s_2) = (b"Ask me no questions, and I'll tell you no lies",
1360151766299025324491939262300636817380452413968031614733084585164 1,
5354638027707905626045156057361096890377811387248394522419069236340)

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

1- Signature verification passes with the present values Sig_Ver(msg1, r1, s1, key, q, p, g) and Sig_Ver(msg1, r1, s1, key, q, p, g)
2- I was able to find the coefficient between k1 and k2 as x=4
3- The following equations can be used for finding the private key if we know the x:
   $s_i = k_i^{-1}(h_i + ar_i) \bmod q$ and
   $s_j = k_j^{-1}(h_j + ar_j) = k_i^{-1}\, x^{-1}(h_j + ar_j) \bmod q.$
   $k_i = s_i^{-1}\,(h_i + ar_i) \bmod q = s_j^{-1}\, x^{-1}(h_j + ar_j) \bmod q.$
   $s_j x(h_i + ar_i) = s_i(h_j + ar_j) \bmod q.$
   $a(s_j r_i x - s_i r_j) = s_i h_j - s_j h_i x \bmod q.$
   $\boldsymbol{a = (s_i h_j - s_j h_i x)(s_j r_i x - s_i r_j)^{-1} \bmod q}$
   a=
   74179633690134148472642049102511301056621710684413575260582871379 6