**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 andexpected find out the corresponding plaintext.You can query the RSA Oracle with any ciphertext , and you will be given the corresponding plaintext . You can send as many queries as you want as long as **.**

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:

609582572245086581955173486556226059889437853770629680858438291721089382181432413886171267314993869601715255

The code is available at Q1.py

1. (**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 . 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.

1. (**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** = 12515043909394803753450411022649854273721537251011107748826811168459680628351391154487041320595006736239332192492236943966523053744476127728797963808151142506595330120621663371518281181204797831707349436558443139355672347825267728879376289677517268609959671235059224994785463608330669494457163250373581380036247652030969481046772013799271268710104487022164865004802864076066974153012125551060906054112920469869045223329577015935824864428612446723942040465300185917923305042033306319809712618872063796904132788285518497999327485929730921202745935936913834577610254298809205575162005025170878200786590751850006857921419

**g** = 2256483143741433163413007675067934542893022968337437312283381964942344365449719628255630752397325376452002398784394008507857025386943645437696558240874471345442532398588406749907930002481624160959132193798842426822193910104962138845873425590946341754334144292886002962901550160578482452138075339294826241799645761655320983735381974177635207208471824667516956679913974643342159550037320378814445802296879470561504511689460916200417902612323039671250567503846175990654512915878143201233050978046269551126178155060158781645062181955781969136435905570787457855530003987887049118699525033120811790739590564684316550493132

**public key (h)** = 1265126138933377994348793193477342224736956600354964713945582205290651827674605003741290400826146165752452701594226002213036650208863340321329798489264160728930653315907521926136642928347549825144026262035747350182493795559385070130959552499813885202334575993642935128132458545523498489490586883187848396314164874056757696154989511633927620869557222556876855999079308839417416012746206040455611002092520255736121673298963050693639916367968280807028975614596114022230524360150581344884219834519025619777858430431159461562871537004523472161672182851052258466610762884570310894027628303901161674783788320479747219000276

And the resulting ciphertext is

**r =** 3813677439444837990381281624769265484071989883494833765363155214071727573627590213038823018054653614040833306533736593789523636716088751609591517852868217052905415751457961942309213803782661174042131067555996860094296315483087375444362454092891960492098796234624392186112659124915872546640723139762874453050592110272036917039293020539724872406856066252779419482651672320132092421939867392668795959155312634804888215300607725584330531720210355201550529764936881761210810883102986464111409096572364185502722477587178710137175828696000683028806920671859797982157383943866111320227830105178421690303627627943337128795446

**t=** 10192033240113377640860169195054315981727514327329008790444130729107056930047299547755150775636237252367979032815426685448329207318153807026080454908281501010744250818034651670583477952735248499512182344163892706808295058861406156976805817052235913385764008189983049947270530103937035152142021836645533913141135755114076443812677194719578205394500861775715274421701692402012349095849491286883920257290062297268751154540379108778881866697419701006074165418163601856726590995982244188091368811214058385356303967475393274055097781937296940874619027724511050397080303621427005403200736696096764013637291006737753794119814

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:

4577933565607691214689963654891646595903581846406907626077086469304820420435450020534356449723013351214400119651700145422949033503144323329091288273353023

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.

1. (**20 pts**) Consider the DSA scheme implemented in the file “DSA.py”. The public parameters and public key are:

**q =** 18462870797958734358460540315802311963744999954506807981508498635091

**p =** 21844102112122237484058484990223222527816981702828279171498143036582716271485474028380542696862193720852272618397503658771128114568430034544311836848132556591324273117839115478343051538427437664722980830771161939139222964707695276957432968033365352302080366315415735532111302710857807281798249043320899027800135122873123243743524724602070457967657285884563858968187732680723369906222214201250288443824722261682828970158731587663585174032887767988219143996717380923998096794060064023264584949115354715211375168860544716843940259887168163262505413440632980952366656691935232538721726450037087263854935179798694999345517

**g =** 13843079639351340920273184714590884400432847093058770970775133079628015343474638985949514224469231316509301786191837239734743524804707156837615319355419215945094865320399756037490734275197507243978890158231379210099367755690209217652326933425758170008835084657241675545571324146202714002127571892258435472678396358353938476569410849475658691697420643000086724156167275855286708191941521213998074404126295230559090196852525498568126029906179168789585152438330622252753643553805877257623433974639379577436808678860489830511416186993204671106346196262903362008285485594747047950971109814842643611103016670841253194356243

**public key - beta =** 6187481213658176498787124123601684091780046690985227386674127034254039365850646655310542241724937514112519192485497669738105144173607992347626869972509174309127140941080651743898030456747633487761927322752193676176314211884662768871783260572354989592156755352437101758031330846064492530779348477298394716501400849788380847680039744807953192006233069850428367974025006391433578254859633968702925514987402010031888483663325943692618870576893826021018783543580318493456251127341437691102522482919743872855098214539426447960934626890138798345418250945885432084267499991534185991486840567366979305573275554091497155603826

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

(message1, , ) = (b"He who laugh last didn't get the joke", 6164572993148268278544315246158794966061243456603081427389792698784, 2412874836775368230194957659405258449579579568340501217618177629780)

(message2, , ) = (b"Ask me no questions, and I'll tell you no lies", 6164572993148268278544315246158794966061243456603081427389792698784, 343379365128270720539597367095485301128970178274104846189598795161)

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* = (*sihj* − *sjhi*)(*r*(*sj* − *si*))−1 mod *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**

1. (**20 pts**) Consider the DSA scheme implemented in the file “DSA.py”. The public parameters and public key are:

**q =** 15141339084211537780798402821468668253233855293250282470707486523729

**p =** 15459352678170194999059797953835943703769299798522640485949251021230061239872933286596281671875036444766767260825161156339142374953144264667175663093532210016977000296281428180052962512096930034626707240943073909429948568647175489641923947055523690662397275499814011659615933313001220733558180164993086472379325887209418439076036830595968948122463542565488458285559269152814846930461678806155717771594791617514000333739836058367191702301817095873715810768950392576601345434651042282496258898798293897916341315693731763534513871295870117294672305447940132333142894162790759196704240972899412016593006223087871357404969

**g =** 3800569625008648766049545537807478639158256666453837543156865205157342453175195338293914518318389932512419197022492193267072466754594620461534567362497841710002599111953091344930343994503431071692400525354528547918075410538790275781900267312641988973075426468087022427855954288858299458927808889518984317490141729401786342725042250941182574740334793901912974170222604015177323368814264989835679407076289974855552414398779625521837257916022552980027627057473062644879659632681204107806120144998907991338913266334321160324651484012752441634140243465730939619242515280714356873699965985363402010686851443396200018800199

**public key - beta =** 13811718194912887731259973687531659017221233072693758339320677556085961091741512534312991319990988012320895125273138799484930424656328618986338233650799555131896857586001490595604365368085682743275712428137943225119715628405892357306029150574584119785832325605674838801154641895745311161271889436502899846458131900988387777254676157672199525938326470244363881227814557082187788046660952433631553517068095734365024876910709029416850114854064043338879940542901936624969303248595208108795751225387203405395739941042570698164719973037261394764330314120509607344408485820133307388882699955010320183318447065675487861322141

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

(message1, , ) = (b"He who laugh last didn't get the joke", 7807207725923213670059456706077357545604668400924354746850607726310, 10137413521818981860558295844142463248736280669671376607939774420169)

(message2, , ) = (b"Ask me no questions, and I'll tell you no lies", 13601517662990253244919392623006368173804524139680316147330845851641, 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:

*si* = *ki*−1(*hi* + *ari*) mod *q* and

*sj* = *kj*−1(*hj* + *arj*) = *ki*−1 x−1(*hj* + *arj*) mod *q*.

*ki* = *si*−1(*hi* + *ari*) mod *q* = *sj*−1x−1(*hj* + *arj*) mod *q*.

*sj*x(*hi* + *ari*) = *si*(*hj* + *arj*) mod *q*.

*a*(*sjrix*− *sirj*) = *sihj* − *sjhix* mod *q*.

***a* = (*sihj* − *sjhix*)(*sjrix*− *sirj*)−1 mod *q***

**a=**

**741796336901341484726420491025113010566217106844135752605828713796**