**Homework #4**

Due date: **11 December 2023**

**Notes**:

* Note that there are five attached files: “RSA\_Oracle\_client.py” for Question 1, “RSA\_OAEP.py” for Question 2, “ElGamal.py” for Questions 3 & 4 and “DSA.py” for Question 5.
* 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 “http://harpoon1.sabanciuniv.edu:9999”. Connect to the server using the *RSA\_Oracle\_Get()* function, and it will send a ciphertext **“ ,** modulus **“**,andpublic key **“e”.**

* You are expected to find out the corresponding plaintext.You can query the RSA Oracle with any ciphertext using the Python function*RSA\_Oracle\_Query()*, and it will send the corresponding plaintext . You can send as many queries as you want as long as **.**
* You should decode your answer into a Unicode string and check it using *RSA\_Oracle\_Checker().*
* You can use the Python code RSA\_Oracle\_client.py to communicate with the server.

**Important Note:** You have to find a mathematical way to find the message “m”. Once you find it, code it then check your answer. Querying the server blindly won’t get you the right answer.

For this question we are not allowed to use RSA\_Oracle\_Query() function with the given ciphertext. To decipher the message, we can pick an arbitrary random number and cipher it using the public key e and N. After that, we can multiply this new number with the given ciphertext.

Let’s say

m = 65536.

m\_c = m^e mod N

C\_ = C\*m\_c mod N

We can decipher C\_ using RSA\_Oracle\_Query() since C != C\_ and get the message M\_.

After getting M\_, we can calculate the inverse of m and use it to get rid of the m.

m\_inv = m^-1 mod N

m\_inv\_pow = m\_inv^e mod N

M = M\_ \* m\_inv\_pow mod N

The above calculation will give us the message m.

**M = Bravo! You found it. Your secret code is 32645**

In the digit form = 156053138494688764261663969923255791456060807758745999627667906341098899958264465653186776286254402261747119157

1. (**20 pts**) Consider the RSA OAEP implementation given in the file “RSA\_OAEP.py”, in which the random number R is an 8-bit unsigned integer. I used the following parameters for encryption:  
     
   **ciphertext** (**c**) = 15563317436145196345966012870951355467518223110264667537181074973436065350566  
   **public** **key** (**e**) = 65537  
   **modulus** (**N**) = 73420032891236901695050447655500861343824713605141822866885089621205131680183

I selected a random four-decimal digit PIN and encrypted it using RSA. Your mission is to find the randomly chosen PIN.

Four decimal digits is not too big for brute force attack.

Also, the randomness for each number is not too big (2^8).

We can conduct a brute force attack by trying all possible combinations of the four digits with all possible randomness. We can encrypt the four digits with all possible randomness and compare the result with the given c. If the result matches, than we find the four digits.

Four decimal digits is between 1000 to 9999. So, there are 9000 different values. Which is approximately 2^13. If we include the randomness, we need to try 2^21 different pairs which is not too big for a brute force attack.

**The four digits are: 1308**

**The random number is: 206**

1. (**20 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** = 20229678282835322453606583744403220194075962461239010550087309021811

**p** = 12515043909394803753450411022649854273721537251011107748826811168459680628351391154487041320595006736239332192492236943966523053744476127728797963808151142506595330120621663371518281181204797831707349436558443139355672347825267728879376289677517268609959671235059224994785463608330669494457163250373581380036247652030969481046772013799271268710104487022164865004802864076066974153012125551060906054112920469869045223329577015935824864428612446723942040465300185917923305042033306319809712618872063796904132788285518497999327485929730921202745935936913834577610254298809205575162005025170878200786590751850006857921419

**g** = 2256483143741433163413007675067934542893022968337437312283381964942344365449719628255630752397325376452002398784394008507857025386943645437696558240874471345442532398588406749907930002481624160959132193798842426822193910104962138845873425590946341754334144292886002962901550160578482452138075339294826241799645761655320983735381974177635207208471824667516956679913974643342159550037320378814445802296879470561504511689460916200417902612323039671250567503846175990654512915878143201233050978046269551126178155060158781645062181955781969136435905570787457855530003987887049118699525033120811790739590564684316550493132

**public key (h)** = 1265126138933377994348793193477342224736956600354964713945582205290651827674605003741290400826146165752452701594226002213036650208863340321329798489264160728930653315907521926136642928347549825144026262035747350182493795559385070130959552499813885202334575993642935128132458545523498489490586883187848396314164874056757696154989511633927620869557222556876855999079308839417416012746206040455611002092520255736121673298963050693639916367968280807028975614596114022230524360150581344884219834519025619777858430431159461562871537004523472161672182851052258466610762884570310894027628303901161674783788320479747219000276

And the resulting ciphertext is

**r =** 3813677439444837990381281624769265484071989883494833765363155214071727573627590213038823018054653614040833306533736593789523636716088751609591517852868217052905415751457961942309213803782661174042131067555996860094296315483087375444362454092891960492098796234624392186112659124915872546640723139762874453050592110272036917039293020539724872406856066252779419482651672320132092421939867392668795959155312634804888215300607725584330531720210355201550529764936881761210810883102986464111409096572364185502722477587178710137175828696000683028806920671859797982157383943866111320227830105178421690303627627943337128795446

**t =** 6879085872532883496679637289827758044388493592192276485018420467127175692447676225327570450845191312409829734608730732636786181351723791499758734679978259974439564271116431478559920406924286698115291434529879801947864012484928324116410562713859998987659607317050575329142948832616264096105332977657905567987590712241261025634719542322903245193802690474499323009626862853070359755324776198222903161231737321083180819484704533141360402185872920868401635784923300080366065601626511550338585716446732854372219141954019480903146819295527105219685774367349234762081116514728907468721241055649461751711410066128218786241602

Can you find the message? (**Note:** The message is a byte object that contains a meaningful English text.)

There is an error in the number random number k generation in the encryption function. There are 65535 possible values for k, but the range of k supposed to be 1 < k < q-1. We can conduct a brute force attack by trying all possible values of k since it is small, 2^16 possible values for k.

After finding the k, we can find the message using the public key (h).

t = h^k \* m mod p

we have h and k.

Find h^-k mod p and multiply it with t (in mod p). The result will be m.

**Message = Be yourself, everyone else is already taken.**

**K = 31659**

1. (**20 pts**) We encrypted two messages, m1 and m2, using the ElGamal encryption algorithm given in “ElGamal.py”, however, it contains a flaw, and we lost m2.

**q** = 1445431254694174381649371259143791311198736690037

**p** = 137248121434045436247980738953059412416367251619167172965225060439638326312552007992983578734870080149141102688002009860722627928048376753275218309927198296531391131491381377746970705292972549293385978940242862964757496679733959578043293370426396437630135799843979374589693726945392682404824784160383287430661

**g** = 127223641921850109909544249881449009944648689040286349526712184078921702602665543540563817762837809423359475544561229778960073396175252439333049143438367080170746166373310913545533812707513022571241268299810387846306038162098727078834162806032355796383642287190219288720676739470587659262303423658215573377024

(b'Believe in the heart of the cards.',

98112636909089823473886804230734608783665151359820285384385184926586779318832342840446756845270685151843520592521030778063107461479185584129724838500026741966009706375181200973944291377753293535599870196345794839828387911579809223830195674821079902123700459948419493000955974605340400274643934795418117953431,

76506200278870980622832162087706397184942731175881073072279653879125374026784231243082249838570209197788703418994598663770222774958590484366297473464547976157101536739056638340401709973910922952987332961258414506877745248599494701005790194262083540626575172771336888597402032923407057219028984697739294234494)

(b'???????????????????????????????????',

98112636909089823473886804230734608783665151359820285384385184926586779318832342840446756845270685151843520592521030778063107461479185584129724838500026741966009706375181200973944291377753293535599870196345794839828387911579809223830195674821079902123700459948419493000955974605340400274643934795418117953431,

95801086901355834240081662719865802187550109851113545620170852280638597493801662857576200633666749663318260607079963837967122188013554434395565196430708343554452720734056250267521097855586180792722796772893530089500987302933561979841152407078582329739116130182358926512269862531407749668332924957717479984854)

Can you recover m2 using the given settings? If yes, demonstrate your work. (**Note:** m2 is a byte object that contains a meaningful English text.)

Since r\_1 == r\_2, the same k was used to encrypt both messages. This gives us the following equations:

b is the private key

B = g^b mod p

r\_1 = r\_2 = g^k mod p

t\_1 = B^k \* m\_1 = g^b\*k \* m\_1 mod p

t\_2 = B^k \* m\_2 = g^b\*k \* m\_2 mod p

We can find g^b\*k by multiplying t\_1 with m\_1 inverse in mod p.

Later, we can find m\_2 by multiplying t\_2 by g^b\*k inverse in mod p.

**M2 = A person can change, at the moment when the person wishes to change.**

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

q = 18055003138821854609936213355788036599433881018536150254303463583193

p = 17695224245226022262215550436146815259393962370271749288321196346958913355063757122216400038699125897137338245645654623180907445775397476914326454182331200843039828753210051963838673399537750764519381124074022003533048362953579747694997421932628050174768037008419023891955638333683910783296320068313502467953549845629364328685168055331330378439460107262672207911384029916731040428600795952248385683448339051326373879623024586381484917048530867998300839452185045027743182645996068845915287513974737094311071485279830178802332884322953485032954055698263286829168380561154757985319675247125962424242568733265799534941009

g = 4789073941777232663925946116548512236454007195930716545844255515671921902088454647562920559586402554819251607533026386568443177012595965432651516494873094284671880587043080168709792729580864399522070440013588701427100770785527321717784068531253489015313171638446034805847845720567691412760307220603939165634874434595948570583948951567783902643539632274510317008676675644324152107083325484901562104857644621121348409411557653041824973063215599539520882871449851513387270613400464314879652836352363637833225350963794362275261801894957372518031031893668151623517523940210995342229628030114190419396207343174070379971035

public key (beta): 1831408160533218510686903726138665932536518466931856989835941853268730468186911958415037229987343935227988816813155415974234360530276380966386586121747340348158553225363319918657949382937198455018294836381584550181800201868806694527418279797492758151769276850910944244395645572497766748854242598561659704665374023326770662512666613356092618904914953512155804252127648818534285831773370510453137952688543495010103660892413395901461238209725480737625047159275781922088076720717434062444236969393756880954396658965471745598003472511293882525516878617801436300794663357187223445935638034452125753926695866508095018852433

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

(message1, , ) = (b'The grass is greener where you water it.', 16472915699323317294511590995572362079752105364898027834238409547851, 959205426763570175260878135902895476834517438518783120550400260096)

(message2, , ) = (b'Sometimes you win, sometimes you learn.' 14333708891393318283285930560430357966366571869986693261749924458661, 9968837339052130339793911929029326353764385041005751577854495398266)

Also, you discovered that . Show how you can find the secret key .

K2 = 3k1 using this fact we can get the following equations.

h1 = H(m1)

r1 = (g^k1 mod p) mod q

s1 = k1^-1(h1 + ar1) mod q

h2 = H(m2)

r2 = (g^k2 mod p) mod q = (g^(3k1) mod p) mod q

s2 = k2^-1(h2 + ar2) mod q = (3k1)^-1(h2 + ar2) mod q

3\*s2 = 3\*(3k1)^-1(h2 + ar2) mod q

= (k1)^-1(h2 + ar2) mod q

Using these equations, we can solve for k1. The explanation and the equations are in the below.

s1\*r2 = k1^-1\*h1\*r2 + k1^-1\*ar1\*r2 mod q

3\*s2\*r1 = k1^-1\*h2\*r1 + k1^-1\*ar2\*r1 mod q

=> subtracting the two equations above

s1\*r2 - 3\*s2\*r1 = k1^-1(h1\*r2 - h2\*r1) mod q

we know h1, h2, r1, r2, s1, s2 and q so we can solve for k1.

After we find k1, we can find a by solving the equation below:

s1\*h2 = k1^-1\*h1\*h2 + k1^-1\*a\*r1\*h2 mod q

3\*s2\*h1 = k1^-1\*h1\*h2 + k1^-1\*a\*r2\*h1 mod q

=> subtracting the two equations above

s1\*h2 - s2\*h1 = k1^-1\*a\*r1\*h2 - k1^-1\*a\*r2\*h1 mod q

= k1^-1\*a(h2\*r1 - h1\*r2) mod q

we know s1, s2, h1, h2, r1, r2, q, and k1 so we can solve for “a”

**k1: 5140023535445352790665837782773385660475477084269771333890682516453**

**k2: 15420070606336058371997513348320156981426431252809314001672047549359**

**a: 2247688824790561241309795396345367052339061811694713858910365226453**