



SECURE COMMUNICATION

**RSA CHALLENGE REPORT**

**By**

**Valery Aime Nundoya (B0099110)**

***School of Informatics***

***Department of Informatics and Engineering***

***Institute of Technology, Blanchardstown***

***Dublin 15.***

**Actual Words**  **Bachelor of Science Computing in**

**Information Technology**

**21/04/2018**

Table of Contents

[Introduction 4](#_Toc6728174)

[challenge Level 0 4](#_Toc6728175)

[challenge Level 1 5](#_Toc6728176)

[Action codes 5](#_Toc6728177)

[Result 6](#_Toc6728178)

[challenge Level 2 7](#_Toc6728179)

[Action codes 7](#_Toc6728180)

[Result 9](#_Toc6728181)

[Challenge Level 3 9](#_Toc6728182)

[Action codes 9](#_Toc6728183)

[Result 10](#_Toc6728184)

[Challenge Level 4 11](#_Toc6728185)

[Action codes 11](#_Toc6728186)

[Result 14](#_Toc6728187)

[Challenge 4.5 14](#_Toc6728188)

[Action 15](#_Toc6728189)

[Result 15](#_Toc6728190)

[Challenge 5 Level 5 16](#_Toc6728191)

[Action codes 16](#_Toc6728192)

[Result 17](#_Toc6728193)

[Challenge 6 18](#_Toc6728194)

[Action codes 18](#_Toc6728195)

[Result 19](#_Toc6728196)

[Challenge 7 20](#_Toc6728197)

[Action codes 20](#_Toc6728198)

[Result 21](#_Toc6728199)

[Challenge 8 22](#_Toc6728200)

[Action Codes 22](#_Toc6728201)

[Results 23](#_Toc6728202)

[Conclusion 24](#_Toc6728203)

[References 25](#_Toc6728204)

[Figure 1 Level 0 5](#_Toc6728243)

[Figure 2 Level1 6](#_Toc6728244)

[Figure 5 Level 1 Result 7](#_Toc6728245)

[Figure 6 Level 2 8](file:///C:\0-SECOND%20SEMESTER\SECURE%20COMMUNICATION\00-PYTHON%20LABS\B00099110%20VALERY%20NUNDOYA%20%20RSA%20CHALLENGE%20LAB.docx#_Toc6728246)

[Figure 8 Result for Level 2 10](#_Toc6728247)

[Figure 9 Level 3 10](#_Toc6728248)

[Figure 11 Provided key to get the values of n, e and d 11](#_Toc6728249)

[Figure 12 Result for Level 4 Challenge 11](#_Toc6728250)

[Figure 13 Level 4 Challenge 12](#_Toc6728251)

[Figure 14 Cipher text for Level 4 12](#_Toc6728252)

[Figure 16 Level 4 Result 15](#_Toc6728253)

[Figure 17 Level 4.5 15](#_Toc6728254)

[Figure 18 Level 4.5 hint 16](#_Toc6728255)

[Figure 19 Level 5 17](#_Toc6728256)

[Figure 21 Result for Level 5 18](#_Toc6728257)

[Figure 22 Level 6 19](#_Toc6728258)

[Figure 24 Level 6 Result 20](#_Toc6728259)

[Figure 25 Level 7 21](#_Toc6728260)

[Figure 26 Level 7 flag 22](#_Toc6728261)

# Introduction

RSA (Rivest-Shamir-Adleman )challenge according to (wikipedia, n.d.) refers to a computational number theory Factoring large integers and cracking cryptography challenge. It is also said that, it is one of the first public key cryptosystems used to secure data transmission.

# challenge Level 0

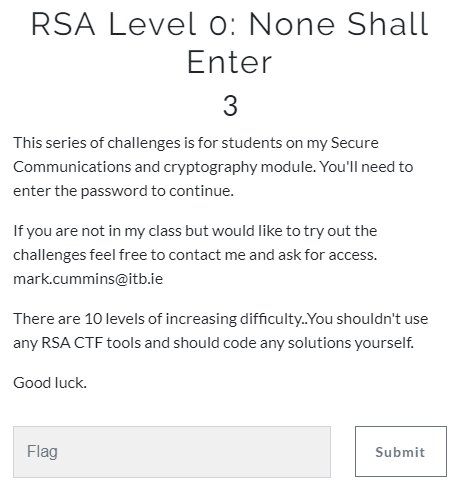


Figure 1 Level 0

In this level 0 I basically entered the password to be able me to see the next challenge which was : letmedothechallenges.

# challenge Level 1

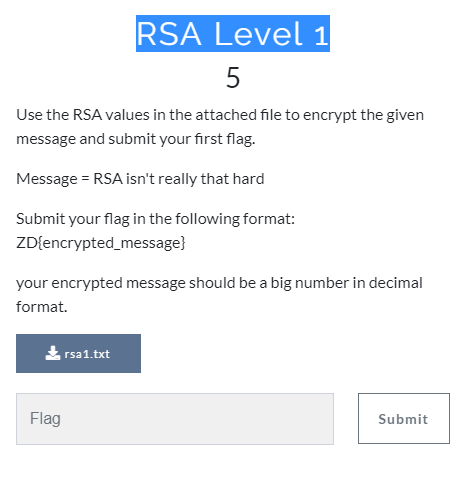


Figure 2 Level1

## Action codes

In this level 1 challenge , a text document code is given to use and encrypt the given message “RSA isn’t that hard” .

Binascii library was imported, which is a python module that contain support functions of number used for encoding.

**import** binascii *#binascii library imported***def** string2int(my\_str): *# method to convert String to integer* **return** int(binascii.hexlify(my\_str), 16) *#return converted String***def** int2string(my\_int): *# method to convert interger to String* **return** binascii.unhexlify(format(my\_int, **"x"**).encode(**"utf-8"**)).decode(**"utf-8"**) *#returning binary ascii code passing in an integer  
  
# -------------------------------------------------------------------------*n = 23516695565660963250242846975094031309572348962900032827958534374248114661507001374384417953124930587796472484525315334716723068326965228898857733318407681656604325744994115789416012096318656034667361976251100005599211469354510367804546831680730445574797161330145320706346512982316782618118878428893337849886890813813050423818145497040676697510093220374542784895778086554812954376689653727580227087363619223145837820593375994747273662064715654881379557354513619477314410917942381406981452545764657853425675230343749326640073923166795823683203941972393206970228647854927797483660176460658959810390117898333516129469397  
e = 65537  
d = 9587600726595591453426898215169101767863399178169979967502694355028996988583633210586039386751682566723132708455252764519220038491664005843242439790264046968625524201298469258242007220372280857992847470031480553726983707671745159488070659256258857978134570602562717609180653377092666963295822401721181836384326336158085408894694549470434424808812412260714422693522311366681659987060925945689943522825747715934700712908720597323076354591388316712970722935035250113120539406041972135508540472211484760814740089404942374666334486855389174327639061106567747152104666795257954039030591097174242386069752606041990644663125  
p = 170436857437540785902894247445629309884819493988198726337160363787266132388801445377172350883259146330710518633323153950488107255453274647690833952071079266615535462115718628529996080297946386916054952930963525522668498855400580516951309863503734146131687670337990358661269686138903141878297721385390421204703  
q = 137978932017559751745702136624874154954496829862527332457067512249687998333117572719846957168595861866495967632464915097378576596911015571165340454225721218087595428364080801400548238088288742249145662369868461078198744980520572785232341389134600070345564258064842348774203427257497319140459851255774165194699  
  
message =**b"RSA isn't really that hard"** *## b before the string to says its a byte string  
## --------------------------------------------------------------------------  
  
## ----- convert message to an int then encrypt ------------*m = string2int(message)  
ciphertext = pow(m, e, n) *## encrypt*print (ciphertext)  
  
*## ----- decrypt cuphertext then convert number back to a string*decrypted = pow(ciphertext, d, n) *# decrypting using the pow decryption method*plaintext = int2string(decrypted) *# decrypting using the pow decryption method*print (plaintext) *# decrypted message converted and pass into plaintext*

## Result

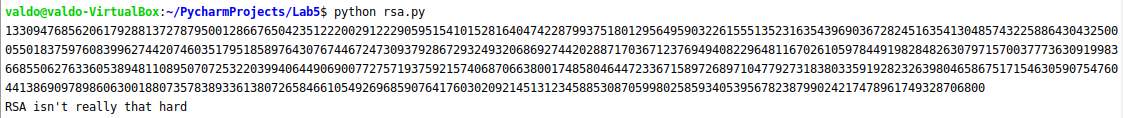


Figure 3 Level 1 Result

# challenge Level 2

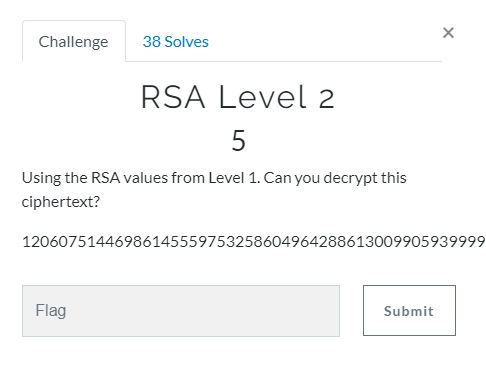


Figure 4 Level 2

## Action codes

This challenge 2 lab is a ciphertext. I e, it is using the RSA values from Level 1 to decrypt the ciphertext.

The Binascii was imported, the string method is define to convert a string to an integer in other to return a converted string .

Passing the given message and the ciphertext2 to be decrypted using pow

Decryption should be in plain text.

*## Commands to generate keys with openssl from commandline.. not part of this python code.  
## openssl genrsa -out mykey.pem  
## openssl rsa -in mykey.pem -pubout > mykey.pub  
## -------------------------------------------------------------------------  
  
## To run type python rsa.py from the commandline (assuming you've python installed***import** binascii *#binascii library imported***def** string2int(my\_str): *# method to convert String to integer* **return** int(binascii.hexlify(my\_str), 16) *#return converted String***def** int2string(my\_int): *# method to convert integer to String* **return** binascii.unhexlify(format(my\_int, **"x"**).encode(**"utf-8"**)).decode(**"utf-8"**) *#returning binary ascii code passing in an integer  
  
# -------------------------------------------------------------------------*n = 23516695565660963250242846975094031309572348962900032827958534374248114661507001374384417953124930587796472484525315334716723068326965228898857733318407681656604325744994115789416012096318656034667361976251100005599211469354510367804546831680730445574797161330145320706346512982316782618118878428893337849886890813813050423818145497040676697510093220374542784895778086554812954376689653727580227087363619223145837820593375994747273662064715654881379557354513619477314410917942381406981452545764657853425675230343749326640073923166795823683203941972393206970228647854927797483660176460658959810390117898333516129469397  
e = 65537  
d = 9587600726595591453426898215169101767863399178169979967502694355028996988583633210586039386751682566723132708455252764519220038491664005843242439790264046968625524201298469258242007220372280857992847470031480553726983707671745159488070659256258857978134570602562717609180653377092666963295822401721181836384326336158085408894694549470434424808812412260714422693522311366681659987060925945689943522825747715934700712908720597323076354591388316712970722935035250113120539406041972135508540472211484760814740089404942374666334486855389174327639061106567747152104666795257954039030591097174242386069752606041990644663125  
p = 170436857437540785902894247445629309884819493988198726337160363787266132388801445377172350883259146330710518633323153950488107255453274647690833952071079266615535462115718628529996080297946386916054952930963525522668498855400580516951309863503734146131687670337990358661269686138903141878297721385390421204703  
q = 137978932017559751745702136624874154954496829862527332457067512249687998333117572719846957168595861866495967632464915097378576596911015571165340454225721218087595428364080801400548238088288742249145662369868461078198744980520572785232341389134600070345564258064842348774203427257497319140459851255774165194699  
  
message =**b"RSA isn't really that hard"** *## b before the string to says its a byte string  
## --------------------------------------------------------------------------  
  
## ----- convert message to an int then encrypt ------------  
#m = string2int(message)  
#ciphertext = pow(m, e, n) ## encrypt  
#print (ciphertext)*ciphertext2=12060751446986145559753258604964288613009905939999612142364090162932784553289274985360758786650150438909964774489580594646537566480131089578622598287608280958826485540308546699201437758196408375494069165098540792161560520821702762571130453590350894456351542803892333747581478520236785140328470857700892653885938706372579150142033036479912415927913687580857324142121399292745144946430853334522298591089627330825281302600321017113877555269059227306217943736817867883165263245345825896646180907782570834990026416074994773400197992710509066779111650705105160806274734336871715369410205954061055784649468877193662056626998  
  
  
*## ----- decrypt cuphertext then convert number back to a string  
#decrypted = pow(ciphertext, d, n) ## decrypt*decrypted2 = pow(ciphertext2, d, n) *## decrypt  
  
#plaintext = int2string(decrypted)*plaintext2 = int2string(decrypted2)  
  
*#print (plaintext)*print (plaintext2)

# Result

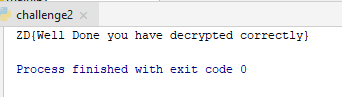
The solution for Level 2 is ZD{Well Done you have decrypted correctly}

Figure 5 Result for Level 2

# Challenge Level 3

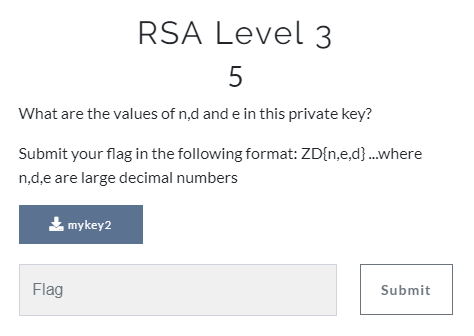


Figure 6 Level 3

## Action codes

In this lab, Crypto library is imported to facilitate the generation of RSA keys using mykeys2 that was provided to get the e, n, d where n, n, d are large decimal numbers.

Private key variable to open mykey2 to be display as a readable file

To get the values of n, e and d, the function will randomly generate a fresh new RSA key object.

**from** Crypto.PublicKey **import** RSA *# rsa import   
  
#Private key vairable to open mykey3 as a readable file*file\_private\_key = open(**"mykey2"**,**"r"**)  
binPrivateKey = RSA.importKey(file\_private\_key)  
  
print(binPrivateKey.n) *# print the value of n*print(**"space"**)  
print(binPrivateKey.e) *# print the value of e*print(**"space"**)  
print(binPrivateKey.d) *# print the value of d*

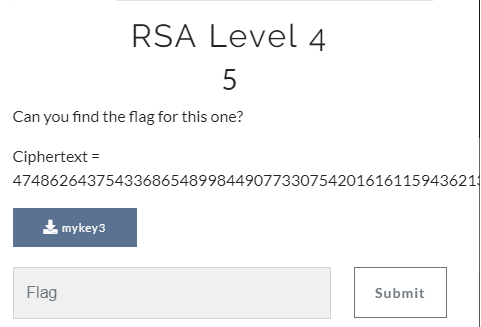


Figure 7 Provided key to get the values of n, e and d

## Result

Figure 3 Result for Level 4 Challenge

# Challenge Level 4



# 

Figure 8 Level 4 Challenge

## Action codes

The task of this lab is to find the flag for the given ciphertext below :

Figure 9 Cipher text for Level 4

In challenge 4, Binascii library is imported once more, along side with the package of Crypto.PublicKey import RSA.

The RSA in this situation is the most widespread and used public key algorithm.

Two functions define of a String and an integer using Binascii to convert between binary and Ascii to return a String for the first function while, the next function will return an integer to be encoded using utf-8

The file private key is passed to open mykey3 as a readable file

Print Values

<https://inventwithpython.com/cracking/chapter24.html>

*## Commands to generate keys with openssl from commandline.. not part of this pythonj code.  
## openssl genrsa -out mykey.pem  
## openssl rsa -in mykey.pem -pubout > mykey.pub  
## -------------------------------------------------------------------------  
## To run type python rsa.py from the commandline (assuming you've pythonh installed  
  
"""  
This code attempts to find the n e d values of the rsa private key  
"""  
#import library***import** binascii  
*# import RSA libary***from** Crypto.PublicKey **import** RSA  
  
  
**def** string2int(my\_str): *# method to convert String to integer* **return** int(binascii.hexlify(my\_str), 16) *#return converted String***def** int2string(my\_int):*# method to convert interger to String* **return** binascii.unhexlify(format(my\_int, **"x"**).encode(**"utf-8"**)).decode(**"utf-8"**) *#returning binary ascii code passing in an integer  
  
# -------------------------------------------------------------------------*n = 23516695565660963250242846975094031309572348962900032827958534374248114661507001374384417953124930587796472484525315334716723068326965228898857733318407681656604325744994115789416012096318656034667361976251100005599211469354510367804546831680730445574797161330145320706346512982316782618118878428893337849886890813813050423818145497040676697510093220374542784895778086554812954376689653727580227087363619223145837820593375994747273662064715654881379557354513619477314410917942381406981452545764657853425675230343749326640073923166795823683203941972393206970228647854927797483660176460658959810390117898333516129469397  
e = 65537  
d = 9587600726595591453426898215169101767863399178169979967502694355028996988583633210586039386751682566723132708455252764519220038491664005843242439790264046968625524201298469258242007220372280857992847470031480553726983707671745159488070659256258857978134570602562717609180653377092666963295822401721181836384326336158085408894694549470434424808812412260714422693522311366681659987060925945689943522825747715934700712908720597323076354591388316712970722935035250113120539406041972135508540472211484760814740089404942374666334486855389174327639061106567747152104666795257954039030591097174242386069752606041990644663125  
p = 170436857437540785902894247445629309884819493988198726337160363787266132388801445377172350883259146330710518633323153950488107255453274647690833952071079266615535462115718628529996080297946386916054952930963525522668498855400580516951309863503734146131687670337990358661269686138903141878297721385390421204703  
q = 137978932017559751745702136624874154954496829862527332457067512249687998333117572719846957168595861866495967632464915097378576596911015571165340454225721218087595428364080801400548238088288742249145662369868461078198744980520572785232341389134600070345564258064842348774203427257497319140459851255774165194699  
  
  
  
file\_private\_key = open(**"mykey3"**,**"r"**) *# given rsa file read into variable*binPrivateKey = RSA.importKey(file\_private\_key) *# keys read from given private rsa private key*print(binPrivateKey.n) *# extracting value of n and printing it out*print(**"space"**)  
print(binPrivateKey.e) *# extracting value of e and printing it out*print(**"space"**)  
print(binPrivateKey.d) *# extracting value of d and printing it out*ciphertext = 474862643754336865489984490773307542016161159436213530034995807183836312346778617047666360854948178434525541089212091928949344492697684657497682106740050084305554758259427768463395264318566101255923490595579348647860471822284428834756812967844672795316325109976652375135659724572710513755433401072885408968307124213606768098411795080747616961236626790699862671834311406129266854138764009952421206625693567227556664511584573464971029270576495696636132292906861410359486612705079004947333371264698887189359265840678094723729950785568382017843975809503403984016678927664449791524785943376314787680072596720311587221852 *## b before the string to says its a byte string  
## --------------------------------------------------------------------------  
  
## ----- convert message to an int then encrypt ------------  
# m = string2int(message)  
# ciphertext = pow(m, e, n) # encrypt  
# print (ciphertext)  
## ----- decrypt cuphertext then convert number back to a string*decrypted = pow(ciphertext, binPrivateKey.d, binPrivateKey.n) *# decrypting using the pow decryption method*plaintext = int2string(decrypted) *# decrypted message converted and pass into plaintext*print (plaintext) *# decrypted message converted and pass into plaintext*

## Result

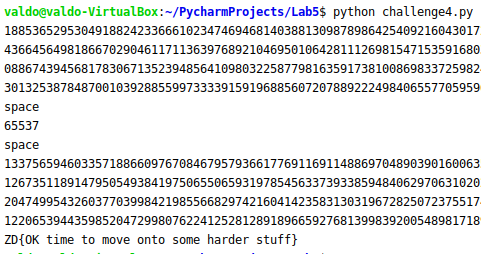


Figure 10 Level 4 Result

# Challenge 4.5

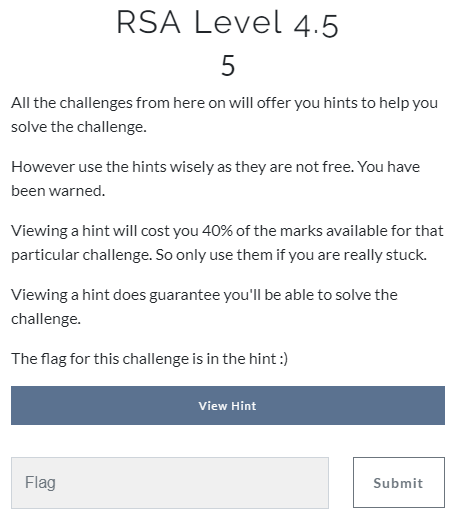


Figure 11 Level 4.5

## Action

This is a straight forward lab; a hint is given to get the flag. The code below is to attempt to find the n e and d values of the rsa private key

## Result

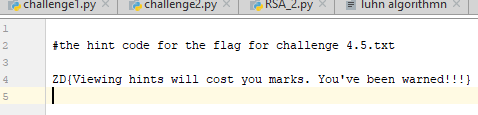


Figure 12 Level 4.5 hint

# Challenge 5 Level 5

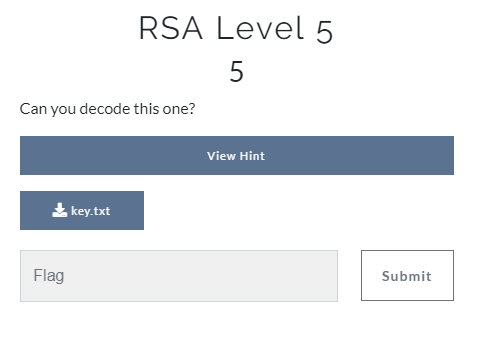


Figure 13 Level 5

## Action codes

Challenge 5 is to decode the provided ket.txt file meanwhile a hint is also provided but viewing will cost points to view.

Binascii library is imported

Two methods are passed one is a string to integer returning a Binascii hexlify input

*## Commands to generate keys with openssl from commandline.. not part of this python code.  
## openssl genrsa -out mykey.pem  
## openssl rsa -in mykey.pem -pubout > mykey.pub  
  
"""  
This code is to decode a code  
"""  
## -------------------------------------------------------------------------  
  
## To run type python rsa.py from the commandline (assuming you've pythonh installed***import** binascii *#import library***def** string2int(my\_str): *# method to convert String to integer* **return** int(binascii.hexlify(my\_str), 16) *#return converted String***def** int2string(my\_int):*# method to convert interger to String* **return** binascii.unhexlify(format(my\_int, **"x"**).encode(**"utf-8"**)).decode(**"utf-8"**)  
  
p = 178303473786440649576050320052579710545803909989892629205383717584926708902831849776048732760539010762697595927799936684517659903020504217647033543451220035557888294087176752636393814444659024479926738965907728305885121533795396510942655193238103461141541226710219917283161689995423704003910208566347008192777  
q = 177704742022668281567275022430710622044625631943464584783356816417678451913814558943536631037762187048467689046234818780661519865062083219860810518179422654892088803969419990517131405785258854547965943290573901682173757368660288706014607783317852866568116668579647747712013357379386510690231133643640946552973  
dp = 1773866135293945458650606659967376768479853354798205505926578632915333539903357890260215965940940766548344180309223167345247940198199013532903029896549971209908039241113252707919629629337895905532939161172648104970277846713071982622558420220505110955098415853869774113380554829745277553298891557215897116769  
dq = 16347740812894503403712423672654444669530920472208797803665993960391586227449959196645900003458629868855939351202369996011540095922292746578891719396733741037038671271672995755493618039875576148888210813721562678209646202536778928064483731718316903925098423712814078626664762373015567892204457096563945050397  
pinv = 52630128849197720680110472750445748362139896176637531822954326517425311680825011848954897216473363639703824050855325538315083014164303720747822395846650319695987688095158291151432519169084976930868513752588843713777689684275678356428411139716649262621622630997667298885499028563774054074332173540631319889597  
qinv = 125496020857340618196277467222560283660900711303887368456426113439542149083570089122371450654613560708642271914201563769861679957096301702280557251661400261492798741353567063566155829049341500619204353629584549900179004920653223519411783060452352992381122206770893349104129009388564667579089451723604618440661  
  
ciphertext = 1388762168166138453533502616535682311951662267796048439821372408514940452694372071133694678245859456415197350818844276387350533386163112380861811751402336635782785232530966339596198327482130002361308263664042358292425147457255505567604410008496199212314343371169549624681927057257495930259138537298520078715021539399084541293912853620643211368004457657505213763568639450075510208834704577814321673791370846192405275347289307606311671974787456421561549138384739624650167355754827475547501943429992429550124101325788544890357396437738989658781888433583926568899431456535167649223812165614500609693719689242432567228419  
  
*## ----- This is to decrypt the ciphertext by converting the number back  
# to a string using the value of dp and p  
  
#This means the m1 = c\*\*dp mod p get the decrypted msg gotten for this link https://en.wikipedia.org/wiki/RSA\_(cryptosystem)#Decryption*decrypted = pow(ciphertext, dp, p) *# decrypting using the pow decryption method*plaintext = int2string(decrypted) *# decrypted message converted and pass into plaintext*print (plaintext) *# decrypted message converted and pass into plaintext*

## Result

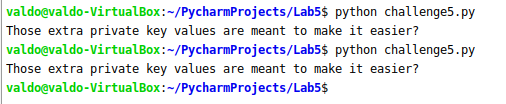


Figure 4 Result for Level 5

# Challenge 6

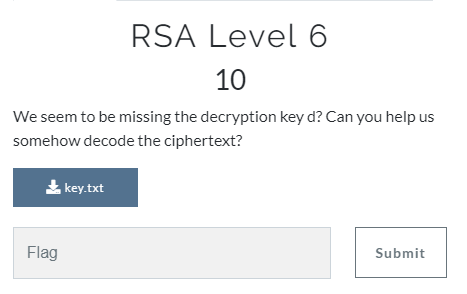


Figure 14 Level 6

## Action codes

Level 6 challenge is to get the missing decryption key d.

Importation of Binascii library, a python module which contains a number of methods

Crypto.PublicKey RSA also imported for crypto as in cryptographic module for python

Functions define

Executing parameters is past with a while loop

To print the result

**import** binascii *#import library***from** Crypto.PublicKey **import** RSA  
  
**def** string2int(my\_str): *# method to convert String to integer* **return** int(binascii.hexlify(my\_str), 16) *#return converted String***def** int2string(my\_int):  
 **return** binascii.unhexlify(format(my\_int, **"x"**).encode(**"utf-8"**)).decode(**"utf-8"**) *#returning binary ascii code passing in an integer  
  
# From wekipeadis search i found modular multiplicative inverse on the following site (https://en.wikipedia.org/wiki/RSA\_(cryptosystem) )  
# Using the mudolar multicative invese, i search on http://www.rosettacode.org/wiki/Modular\_inverse#Python  
# THen i got the function below***def** extended\_gcd(aa, bb):  
 lastremainder, remainder = abs(aa), abs(bb)  
 x, lastx, y, lasty = 0, 1, 1, 0  
 **while** remainder:  
 lastremainder, (quotient, remainder) = remainder, divmod(lastremainder, remainder)  
 x, lastx = lastx - quotient\*x, x  
 y, lasty = lasty - quotient\*y, y  
 **return** lastremainder, lastx \* (-1 **if** aa < 0 **else** 1), lasty \* (-1 **if** bb < 0 **else** 1)  
  
  
**def** modinv(a, m):  
 g, x, y = extended\_gcd(a, m)  
 **if** g != 1:  
 **raise** ValueError  
 **return** x % m  
  
e = 65537  
p = 163598797232837275790583032413921422452851861145478369331976309880028992955089558380171554447759405365296693377570783300198791468861355639873166150884714034914366548252757855530548966926710596087588892893653952147784119788340592861717511574050564549916735627066568966135368285851889401719649796310308064172229  
q = 151928351783926490385254692544226090032004315756120674902384041799040568083955129227360764179393042678005292005933989750269377019057534023167675372696224003953154715102625798599561576746593076228704448522848509650863715575134525964992439285085243915010868628145127710442853766119688772555932018349278733467937  
n = p\*q  
d = modinv(e,( q-1)\*(p-1))  
  
  
*#given cipher in key2*ciphertext = 4413233431418367729487001191499320110908628864393005850336194538378846901872012263024060279733910394528568658924541767014298273106072428208428621362441660742168169457839232452898840402021800460905562638079257404470183053387353849960252811956727755974787563684430128654542847575219444418360279725423441999278619584162289488016498634231451443666882615379215688913514242136494373656647328276909398980200846880640231426382657437148137610018777974884800967755913109702229247523206388812041488414941125272083962209616158810973532091497979384180936871075352614021504627549173686729322478688708849605857667792183339692021980  
  
*#using wikipeadia*decrypted = pow(ciphertext, d, n) *# decrypting using the pow decryption method*plaintext = int2string(decrypted) *# decrypted message converted and pass into plaintext*print (plaintext) *# decrypted message converted and pass into plaintext*

## Result



Figure 5 Level 6 Result

# Challenge 7

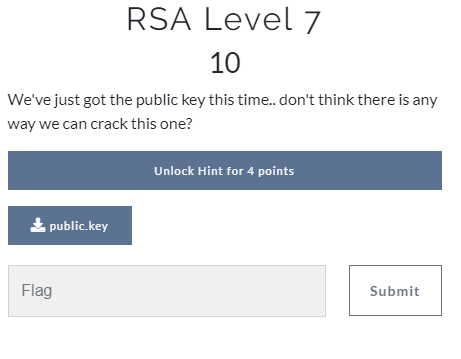


Figure 15 Level 7

## Action codes

Things become more and more difficult . At this level, to get the flag, there a hint that will cost 4 points.

The importation of the library of Binascii and RSA

Declared the method of converting string to into integer etc

*#import library***import** binascii  
*#import rsa library***from** Crypto.PublicKey **import** RSA  
  
*#method to convert string to int***def** string2int(my\_str): *#return converted String* **return** int(binascii.hexlify(my\_str), 16)  
  
*# method to convert interger to String***def** int2string(my\_int):  
 **return** binascii.unhexlify(format(my\_int, **"x"**).encode(**"utf-8"**)).decode(**"utf-8"**) *#returning binary ascii code passing in an integer  
  
# From wekipeadis search i found modular multiplicative inverse on the following site (https://en.wikipedia.org/wiki/RSA\_(cryptosystem) )  
# Using the mudolar multicative invese, i search on http://www.rosettacode.org/wiki/Modular\_inverse#Python  
# THen i got the function below***def** extended\_gcd(aa, bb):  
 lastremainder, remainder = abs(aa), abs(bb)  
 x, lastx, y, lasty = 0, 1, 1, 0  
 **while** remainder:  
 lastremainder, (quotient, remainder) = remainder, divmod(lastremainder, remainder)  
 x, lastx = lastx - quotient\*x, x  
 y, lasty = lasty - quotient\*y, y  
 **return** lastremainder, lastx \* (-1 **if** aa < 0 **else** 1), lasty \* (-1 **if** bb < 0 **else** 1)  
  
**def** modinv(a, m):  
 g, x, y = extended\_gcd(a, m)  
 **if** g != 1:  
 **raise** ValueError  
 **return** x % m  
  
*#http://factordb.com which was given in the lecture, to get the values of P  
#d value was taken fromchange6*n = 79832181757332818552764610761349592984614744432279135328398999801627880283610900361281249973175805069916210179560506497075132524902086881120372213626641879468491936860976686933630869673826972619938321951599146744807653301076026577949579618331502776303983485566046485431039541708467141408260220098592761245010678592347501894176269580510459729633673468068467144199744563731826362102608811033400887813754780282628099443490170016087838606998017490456601315802448567772411623826281747245660954245413781519794295336197555688543537992197142258053220453757666537840276416475602759374950715283890232230741542737319569819793988431443  
e = 65537  
q = 25478326064937419292200172136399497719081842914528228316455906211693118321971399936004729134841162974144246271486439695786036588117424611881955950996219646807378822278285638261582099108339438949573034101215141156156408742843820048066830863814362379885720395082318462850002901605689761876319151147352730090957556940842144299887394678743607766937828094478336401159449035878306853716216548374273462386508307367713112073004011383418967894930554067582453248981022011922883374442736848045920676341361871231787163441467533076890081721882179369168787287724769642665399992556052144845878600126283968890273067575342061776244939  
p = 3133337  
d = modinv(e,( q-1)\*(p-1))  
  
Ciphertext = 877047627503964563527859854056241853286548710266261291942543955818132370489959838496983429954434494528178229313135354793125902041844995518092695073588272773865176510386504459109444540504995243455296652458363596632448945407597570368304177404561607143991631472612686460090955582314803404185085391881900665937993904325795901688452399415391744151647251408176477627720933717024380735888111455809609800839992904182591275652616244755461341372866557636825262065485442416189938154309976219500988259186981644426083447522183242945513870008042818029602927271842718324310884266107435333212981162347887454715321088536179467180247805306  
  
decrypted = pow(Ciphertext, d, n) *decrypting using the pow decryption method*

plaintext = int2string(decrypted) *# decrypted message converted and pass into plaintext*

print (plaintext)

## Result



Figure 16 Level 7 flag

# Challenge 8

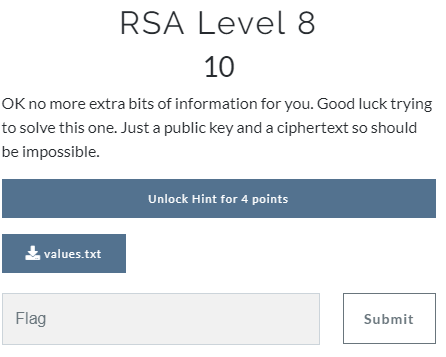


Figure 17 Level 8

## Action Codes

Challenge 8 happens to be one of the tough once. This lab is to try and look for a public key and which a ciphertext should be impossible.

The Binascii library is imported, RSA and that of gypy2 as well

A method to convert string to integer and vice versa is declared

*"""  
This code attempts to get the public key and a ciphertext should he impossible  
"""***import** binascii *#import binascii library***from** Crypto.PublicKey **import** RSA *#import rsa library***import** gmpy2 *#import gmpy2***def** string2int(my\_str): *#method to convert string to integer* **return** int(binascii.hexlify(my\_str), 16) *#return converted String***def** int2string(my\_int):  
 **return** binascii.unhexlify(format(my\_int, **"x"**).encode(**"utf-8"**)).decode(**"utf-8"**) *#returning binary ascii code passing in an integer  
  
# a while loop method to decrypt the cipher***def** extended\_gcd(aa, bb):  
 lastremainder, remainder = abs(aa), abs(bb)  
 x, lastx, y, lasty = 0, 1, 1, 0  
 **while** remainder:  
 lastremainder, (quotient, remainder) = remainder, divmod(lastremainder, remainder)  
 x, lastx = lastx - quotient\*x, x  
 y, lasty = lasty - quotient\*y, y  
 **return** lastremainder, lastx \* (-1 **if** aa < 0 **else** 1), lasty \* (-1 **if** bb < 0 **else** 1)  
  
  
**def** modinv(a, m):  
 g, x, y = extended\_gcd(a, m)  
 **if** g != 1:  
 **raise** ValueError  
 **return** x % m  
  
  
**def** rsa\_unpadded\_message\_attack(ct, modulus, exponent=3): *# rsa attack tool for decrypting* **while True**:  
 rec, e = gmpy2.iroot(ct, exponent)  
 **if** e:  
 **break** ct += modulus  
  
 **return** gmpy2.long\_to\_bytes(rec)  
  
  
n= 23516695565660963250242846975094031309572348962900032827958534374248114661507001374384417953124930587796472484525315334716723068326965228898857733318407681656604325744994115789416012096318656034667361976251100005599211469354510367804546831680730445574797161330145320706346512982316782618118878428893337849886890813813050423818145497040676697510093220374542784895778086554812954376689653727580227087363619223145837820593375994747273662064715654881379557354513619477314410917942381406981452545764657853425675230343749326640073923166795823683203941972393206970228647854927797483660176460658959810390117898333516129469397  
e= 3  
  
ciphertext = 145069245024457407970388457302568525045688441508350620445553303097210529802020156842534271527464635050860748816803790910853366771838992303776518246009397475087259557220229739272919078824096942593663260736405547321937692016524108920147672998393440513476061602816076372323775207700936797148289812069641665092971298180210327453380160362030493  
  
decrypted = rsa\_unpadded\_message\_attack(ciphertext,n,e) *# decrypting using the pow decryption method*plaintext = int2string(decrypted) *# decrypted message converted and pass into plaintext*print (plaintext) *# decrypted message converted and pass into plaintext*

## Results

# Conclusion

Going through these python challenge RSA labs has not been easy but in other word, it was a very good learning process. Some of the challenges were a little bit easier at the start with while some detailed and clear explanation before graduating to tough stuff. I am delighted for taking part in this challenge as it has thought a lot on how RSA challenge works even though I couldn’t complete the last parts.

# References

*wikipedia*. (n.d.). Retrieved from https://en.wikipedia.org: https://en.wikipedia.org/wiki/RSA\_Factoring\_Challenge

[**https://www.dlitz.net/software/pycrypto/api/current/Crypto.PublicKey.RSA-module.html**](https://www.dlitz.net/software/pycrypto/api/current/Crypto.PublicKey.RSA-module.html)

<https://www.dotnetperls.com/rot13-python?fbclid=IwAR2wAFF7IrvimM696qOFydbwmbIcjLqMWyfz1aeIJE_x0-n0kQ5gZDOWzzI>

<http://www.asciitable.com/?fbclid=IwAR0y9hDjCwxmvtVGQtuOQFePtSnqPpQsZgR_woAm0sjG1YBzV38R44ig1SI>

http://installion.co.uk/kali/kali/main/p/python-gmpy2-common/install/index.html