Q1) Since server is returning a message for any cipher text, we can manipulate this weakness as we have the original cipher.

Assume we multiplied our original cipher with a constant value (const) and sent to the server. In this case, returned message (m2) will be related with our hidden original message (m) as following way:

$C1 = M1^e \bmod(n)$

$C2 = (C1*const^e) \bmod(n)$

$M2 = C2^d \bmod(n)$

$M2 = C1^d * const^{ed} \bmod(n)$

$M2 = M1^{ed} * const^{ed} \bmod(n)$ where $ed = 1$

$M2 = M1 * const \bmod(n)$

$M1 = M2 * const^{-1} \bmod(n)$

As it is clear in the above, we can find the original message from response of server.

```
temp = 5
temp_inv = modinv(temp, N)
c_ = (pow(temp,e,N)*c)%N
m_ = RSA_Oracle_Query(c_)
m = (m_ * temp_inv) % N
print("My message is following:", m.to_bytes((m.bit_length() + 7) // 8, byteorder='big'))

RSA_Oracle_Checker(m.to_bytes((m.bit_length() + 7) // 8, byteorder='big').decode("UTF-8"))
```

My Output:

```
My message is following: b'Bravo! You find it. Your secret code is 40963'
Congrats
```

Q2) We can use brute force easily as the message range is small (only 4 digit PIN) and the R range is very small between 128 and 256.

```
myflag = True
for m in range(1000, 10000):
    if (myflag):
        for R in range(2**(k0-1), 2**k0):
            c_ = RSA_OAEP_Enc(m, e, N, R)
            if c == c_:
                print("You have found the PIN = {} with the following R = {}.".format(m, R))
                myflag = False
                PIN = m
                break
    else:
            break
RSA_OAEP_Checker(PIN)
```

My Output:

```
{'c': 53823407077027493744945084449899311224454666585495292549709035871194748531061, 'N':
75912732707060243642078909648401302780483043992228012220203806825283170905549, 'e': 65537}
You have found the PIN = 5120 with the following R = 185.
Congrats
```

Q3) Since q is small, I can just use brute force again to find k. Each time I can check whether we have same r value for given k, if they are same than I can stop my search.

$$r \leftarrow g^k \bmod p$$

```
for k in range(2,q-1):
    if pow(g,k,p) == r:
        print("My k:", k)
        break
temp = pow(h,k,p)
temp_inv = modinv(temp, p)
m = (t*temp_inv)%p
print("My message is following: ",m.to_bytes((m.bit_length() + 7) // 8, byteorder='big').decode("UTF-8"))
```

My Output:

```
My k: 64278
My message is following:  I am gonna make him an offer he cannot refuse
```

Q4) Implementation has a huge fatal error, we can observe we have two same r values for two different encryptions. Thus, we can say that same k value is used for encryption of two different messages. Therefore, I used following formula which is studied in lecture:

$$m_2 = (t_2 m_1)/t_1 (mod\, p)$$

```
if (r1 == r2):
    print("Same k is used for encryption which is a fatal!")
m1_int = int.from_bytes(m1, byteorder='big')
t1_inv = modinv(t1, p)
m2 = (t2*m1_int*t1_inv)%p
print("My message is following:", m2.to_bytes((m2.bit_length() + 7) // 8, byteorder='big').decode("UTF-8"))
```

My Output:

```
Same k is used for encryption which is a fatal!
My message is following: Well, it was more like a command, no was not an option!
```

Q5) Again we have same r values for two different encryptions which results with same alpha values in the encryptions. I used following formula which is studied in lecture:

$$a = (s_i h_j - s_j h_i)(r(s_j - s_i))^{-1} \bmod q$$

```
shake = SHAKE128.new(m1)
h1 = int.from_bytes(shake.read(q.bit_length()//8), byteorder='big')
shake = SHAKE128.new(m2)
h2 = int.from_bytes(shake.read(q.bit_length()//8), byteorder='big')
temp = modinv(r1*(s1-s2), q)
a = ((s2*h1 - s1*h2)*temp) % q
print("My secret key a: ", a)
```

I also wanted to verify my key values at the end:

```
print("Lets verify this is a true secret alpha key value!")
s2_inv = modinv(s2, q)
k = (s2_inv*(h2+ a*r1))%q
print("My k value: ", k)
r, s = Sig_Gen(m1, a, k, q, p, g)
if Sig_Ver(m1, r, s, beta, q, p, g):
    print("signature verifies:) ")
else:
    print("invalid signature:( ")
```

My Output:

```
Same k is used for encryption which is a fatal!
My secret key a:  180114935909579198431966542725302564519161305719138984175086511137437
Lets verify this is a true secret alpha key value!
My k value:  14519395415119252436332027861991089696243713134576320901515642148145
signature verifies:)
```

Q6) This time we do not have same r values, but hint implies that our k values are not independent from each other. Then, we can use a brute force to find a proper x value as it usually has small value. I used following formula which is studied in lecture:

$$a = (s_i h_j - s_j h_i x)(s_j r_i x - s_i r_j)^{-1} \bmod q$$

```python
if (r1 == r2):
    print("Same k is used for encryption which is a fatal!")
shake = SHAKE128.new(m1)
h1 = int.from_bytes(shake.read(q.bit_length()//8), byteorder='big')
shake = SHAKE128.new(m2)
h2 = int.from_bytes(shake.read(q.bit_length()//8), byteorder='big')
x = 1
while True:
    temp = modinv(abs(s2*r1*x - s1*r2), q)
    a = ((s1*h2 - s2*h1*x)*temp)%q
    if pow(g,a,p) == beta:
        print("We have found the x:",x)
        print("Corresponding secret key a:", a)
        break
    else:
        x += 1
```

I also wanted to verify my key values at the end:

```python
print("Lets verify this is a true secret alpha key value!")
s2_inv = modinv(s2, q)
k = (s2_inv*(h2+ a*r1))%q
print("My k value: ", k)
r, s = Sig_Gen(m1, a, k, q, p, g)
if Sig_Ver(m1, r, s, beta, q, p, g):
    print("signature verifies:) ")
else:
    print("invalid signature:( ")
```

My Output:

```
We have found the x: 63
Corresponding secret key a: 66568624500090235129890566130399211243633217014
Lets verify this is a true secret alpha key value!
My k value:  518228419456628617464604756290194409760151243411
signature verifies:)
```

CODES

Q1)

```python
def RSA_Oracle_Get():
  response = requests.get('{}/{}/{}'.format(API_URL, "RSA_Oracle", my_id))
  c, N, e = 0,0,0
  if response.ok:
    res = response.json()
    print(res)
    return res['c'], res['N'], res['e']
  else:
    print(response.json())

def RSA_Oracle_Query(c_):
  response = requests.get('{}/{}/{}/{}'.format(API_URL, "RSA_Oracle_Query", my_id, c_))
  print(response.json())
  m_= ""
  if response.ok:   m_ = (response.json()['m_'])
  else: print(response)
  return m_

def RSA_Oracle_Checker(m):
  response = requests.put('{}/{}/{}/{}'.format(API_URL, "RSA_Oracle_Checker", my_id, m))
  print(response.json())

#get the parameters
c, N, e = RSA_Oracle_Get()
# #choose a ciphertext and get the corresponding plaintext
# m_ = RSA_Oracle_Query(c_)
# #Calculte m using m_
# RSA_Oracle_Checker(m) #m should be string
temp = 5
temp_inv = modinv(temp, N)
c_ = (pow(temp,e,N)*c)%N
m_ = RSA_Oracle_Query(c_)
m = (m_ * temp_inv) % N
print("My message is following:", m.to_bytes((m.bit_length() + 7) // 8, byteorder='big'))
RSA_Oracle_Checker(m.to_bytes((m.bit_length() + 7) // 8, byteorder='big').decode("UTF-8"))
```

Q2)

```python
from RSA_OAEP import *
import math
import timeit
import random
import sympy
import warnings
import requests
from Crypto.Hash import SHA3_256
from Crypto.Hash import SHA3_384
from Crypto.Hash import SHA3_512
from Crypto.Hash import SHAKE128, SHAKE256

API_URL = 'http://cryptlygos.pythonanywhere.com'

my_id = 25331   ## Change this to your ID number

def RSA_OAEP_Get():
  response = requests.get('{}/{}/{}'.format(API_URL, "RSA_OAEP", my_id ))
  c, N, e = 0,0,0
  if response.ok:
    res = response.json()
    print(res)
    return res['c'], res['N'], res['e']
  else:
    print(response.json())
    return c, N, e

def RSA_OAEP_Checker(PIN_):
  # Client sends PIN_
  response = requests.put('{}/{}/{}/{}'.format(API_URL, "RSA_OAEP", my_id, PIN_))
  print(response.json())

#get the parameters
c, N, e = RSA_OAEP_Get()

# #Calculate the PIN_ and check your answer
# RSA_OAEP_Checker(PIN_)

myflag = True
for m in range(1000, 10000):
    if (myflag):
        for R in range(2**(k0-1), 2**k0):
            c_ = RSA_OAEP_Enc(m, e, N, R)
            if c == c_:
                print("You have found the PIN = {} with the following R = {}.".format(m, R))
                myflag = False
                PIN = m
                break
    else:
            break
RSA_OAEP_Checker(PIN)
```

Ege Erdogan 25331 (*ALL CODES AT THE END OF PDF*)

Q3)

```python
# -*- coding: utf-8 -*-
"""
Created on Sun Dec  5 19:28:20 2021

@author: user
"""
from ElGamal import *

q = 21951366550493000718261853105201996539940614036945856492989212434043
p = 190885173558311304099347483787798059053027056419653641599421250901016179997530270181896679835452891862852302944186416050368252406
g = 110874950916035308201558971318404732618584098845349297851209369703503765983404428968835746417446964796284726243423592721516375391
h = 987320377917824491735857382210871486622556457509380838121352205509962690504857471837707984399183951038894244713384696768128945681
r = 132391654622962474731982860849733838649963782531365577404295217191440638318217291777993993059508484290101645915425785626840239462
t = 292804521838291345744368414401211925215870260693565040608272290904375022206353595256844539924654933868653590583751336136243235231


for k in range(2,q-1):
    if pow(g,k,p) == r:
        print("My k:", k)
        break
temp = pow(h,k,p)
temp_inv = modinv(temp, p)
m = (t*temp_inv)%p
print("My message is following: ",m.to_bytes((m.bit_length() + 7) // 8, byteorder='big').decode("UTF-8"))
```

Q4)

```python
# -*- coding: utf-8 -*-
"""
Created on Sun Dec  5 22:56:53 2021

@author: user
"""

from ElGamal import *

q = 1367176787662174613885987459588219879372220953507
p = 123673323891242867177514096819580236465497274063458984325241055844011153224145054858864521371774649105018822971116982650715
g = 404621221188721819453789213524870729395169520186627356860554846147313773002412967932741940153306778745619138622570323493191
m1 = b'I am gonna make him an offer he cannot refuse'
r1 = 358136611273315274693584000616023624685848884055947523542303622721572499086663124962197759732570128299538504165559471799420
t1 = 436272242181157972282892494759210329070342204604923561118955039360828544181682504164970729413474719947437279732546423255585

r2 = 358136611273315274693584000616023624685848884055947523542303622721572499086663124962197759732570128299538504165559471799420
t2 = 595681361927820114080092427206809055456490929597996245063065651974043047750494240739299340959992508520364723319385101864 70

if (r1 == r2):
    print("Same k is used for encryption which is a fatal!")
m1_int = int.from_bytes(m1, byteorder='big')
t1_inv = modinv(t1, p)
m2 = (t2*m1_int*t1_inv)%p
print("My message is following:", m2.to_bytes((m2.bit_length() + 7) // 8, byteorder='big').decode("UTF-8"))
```

Q5)

```python
# -*- coding: utf-8 -*-
"""
Created on Sun Dec  5 23:07:29 2021

@author: user
"""


from DSA import *

q = 21050461915163064005698472752818467960484664222419461240422905587329
p = 1663500126842477024836249602087898279485597304272712311027621822136311524853212165692699849532442288420
g = 4093162097966074909687347020209828087793387673878863126522370015312513519384142780904183350879136499577
beta = 3330001424503044325932197676136127266689391995091913015672597381088363195443636436214100590508995319

m1 = b'Asking questions during the lectures helps you understand Crypto'
r1 = 260444855760506318805841590364189311211267498403457607938240440795
s1 = 15045429964567421250403275656320025283600046882519690784113588548158

m2 = b'Keep your friends close, but your enemies closer'
r2 = 260444855760506318805841590364189311211267498403457607938240440795
s2 = 14016151436550334193141059702675072658308100333231844563375725796770

if (r1 == r2):
    print("Same k is used for encryption which is a fatal!")

shake = SHAKE128.new(m1)
h1 = int.from_bytes(shake.read(q.bit_length()//8), byteorder='big')
shake = SHAKE128.new(m2)
h2 = int.from_bytes(shake.read(q.bit_length()//8), byteorder='big')
temp = modinv(r1*(s1-s2), q)
a = ((s2*h1 - s1*h2)*temp) % q
print("My secret key a: ", a)


print("Lets verify this is a true secret alpha key value!")
s2_inv = modinv(s2, q)
k = (s2_inv*(h2+ a*r1))%q
print("My k value: ", k)
r, s = Sig_Gen(m1, a, k, q, p, g)
if Sig_Ver(m1, r, s, beta, q, p, g):
    print("signature verifies:) ")
else:
    print("invalid signature:( ")
```

Q6)

```python
# -*- coding: utf-8 -*-
"""
Created on Sun Dec  5 23:23:16 2021

@author: user
"""

from DSA import *

q = 1274928665248456750459255476142268320222010991943
p = 943990828777386403563448350936338517422268109465480581675941066095993041014833761
g = 747576130488870932097416342282284259029485722229656838929667828296542988007917890
beta = 93910788220122222642484838539579554500745218470968665334596813695469448862350

m1 = b'Erkay hoca wish that you did learn a lot in the Cryptography course'
r1 = 780456265196245442017019073827244628033034896446
s1 = 214154189471546244965139202160125045302874348377

m2 = b'Who will win the 2021 F1 championship, Max or Lewis?'
r2 = 927294142715241205623350780659879368622965215767
s2 = 151110642214296558517943730901561426792280910589


if (r1 == r2):
    print("Same k is used for encryption which is a fatal!")
shake = SHAKE128.new(m1)
h1 = int.from_bytes(shake.read(q.bit_length()//8), byteorder='big')
shake = SHAKE128.new(m2)
h2 = int.from_bytes(shake.read(q.bit_length()//8), byteorder='big')
x = 1
while True:
    temp = modinv(abs(s2*r1*x - s1*r2), q)
    a = ((s1*h2 - s2*h1*x)*temp)%q
    if pow(g,a,p) == beta:
        print("We have found the x:",x)
        print("Corresponding secret key a:", a)
        break
    else:
        x += 1

print("Lets verify this is a true secret alpha key value!")
s2_inv = modinv(s2, q)
k = (s2_inv*(h2+ a*r1))%q
print("My k value: ", k)
r, s = Sig_Gen(m1, a, k, q, p, g)
if Sig_Ver(m1, r, s, beta, q, p, g):
    print("signature verifies:) ")
else:
    print("invalid signature:( ")
```