<u>Formative Assignment</u>

<u>Exercise 1:</u>
Answer:
a) 4^13 = 4^10 * 4^3= (1) * 64 (MOD 11), 64 – 55 = <u>9 MOD 11</u>
b) 3^3 = 27 = 27 – 22 = <u>5 MOD 11</u>
c) 5^12 = 5^10 * 5^2 = 1 * 25 = 25 – 22 = <u>3 MOD 11</u>
d)10^-10 = 1/(10^10) = 1/1(MOD 11) = <u>1 MOD 11</u>
Further Explanation:
Fermat’s little theorem is Euler’s Theorum for prime numbers, the result of these exercises is gained by reducing these large number using said algorithm as 11 is prime

<u>Exercise 2:</u>
Answer:
(a) p = 7, q = 11, e = 67, M = 75
N = p * q = 7 * 11 = 77
phi(N) = (p – 1)*(q - 1) = 6 * 10 = 60
gcd(e, phi) = alpha * e + beta * phi = 1
gcd(e, phi) = -17 * e + 19 * phi = 1 (MOD 60)
d = e^-1 = -17 = 43 (MOD 60)
SK = d = 43

c = m^e mod N
c = 75^67 MOD 77 = (75^60)* (75^7) =
Using Euler’s you get: 1 * 75^7 = 75^7
75 – 77 = -2 hence (-2^7) = -128 (MOD 77) = - 51 (MOD 77)= 26 (MOD 77)
Ciph = 26

p = c^d mod N
p = 26^43 MOD 77 = (13 * 2)^43 (MOD 77) = 13^43 * 2^43 (MOD 77) =
-64^43 * 2^43 (MOD 77)= -8^86 * 2^43 (MOD 77) =
-8^26 * -8^60 * 2^43 (MOD 77) =
Using Euler’s you get: -8^26 * 2^43 (MOD 77) =
-(2^3)^26 * 2^43 (MOD 77) = -2^78 * 2^43 (MOD 77) = -2^121 (MOD 77)
Using Euler’s twice you get: -2 ^ 1(MOD 77) = -2 (MOD 77) = 75 = M
Plain= 75

(b) p = 5, q = 13, e = 35, M = 54
N = p * q = 5 * 13 = 65
PK(N, e) = (65, 35)
phi(N) = (p – 1)*(q - 1) = 4 * 12 = 48
gcd(e, phi) = alpha * e + beta * phi = 1
gcd(e, phi) = -8 * 35 + 11 * 48 = 1, Hence
35^-1 (MOD 48) = 11 (MOD 48)

d = e^1 MOD phi, Hence
d = 11 (MOD 48)
SK = d = 11 (MOD 48)

c = m^e mod N
c = 54^35 MOD 65 = (2^35) * (27^35) (MOD 65) =
(2^35) * (3^3)^35(MOD 65) = (2^35) * 3^105 (MOD 65)
Using Euler's twice you get: (2^35) * (3^9) (MOD 65) =
2^35 * 3 * (3^4) * (3 ^ 4) (MOD 65) =
2^35 * 3 * 81 * 81 (MOD 65)
Use the modulus two times to output:
2^35 * 3 * 16 * 16 (MOD 65) = 2^35 * 3 * 2^4 * 2^4 (MOD 65) =
2^35 * 3 * 2^8 (MOD 65) = 2^35 * 3 * 256 (MOD 65) =
256 – 195 (65 * 3) = 61
2^35 * 3 * 61 (MOD 65) = 2^35 * 183 (MOD 65) = 183 – 130 (65 * 2) = 53
2^35 * 53 (MOD 65) = (2^7)^5 * 53 (MOD 65) = 128 = -2 (MOD 65) =
53 = -12 (MOD 65) = (-2)^5 * -12 (MOD 65) = (-2) ^ 5 = -32
-32 * -12 = 320 + 64 = 384 (MOD 65) = 384 – 390 (65 * 6) = - 6 (MOD 65) =
65 – 6 = 59 (MOD 65)
C = 59 (MOD 65)
Ciph = 59

p = c^d mod N
p = 59^11 MOD 65 =
(-6)^11 (MOD 65) = (-2 * 3)^11 (MOD 65) = -2^11 * 3^11 (MOD 65) =
-2^3 * 2^8 * 3^3 * 3^4 * 3^4 (MOD 65) = -8 * 256 * 27 * 81 * 81 (MOD 65) =
-8 * 61 * 27 * 16 * 16 (MOD 65) = -8 * -4 * 27 * 16 * 16 (MOD 65) =
32 * 27 * 16 * 16 (MOD 65) = 2^13 * 3^3 (MOD 65) =
2^8 * 2^5 * 3^3 (MOD 65) = 16 * 2^5 * 3^3 (MOD 65) =
2^4 * 2^5 = 2^9 * 3^3 (MOD 65) = 512 – 520 = -8 * 3^3 (MOD 65) =
-8 * 27 = 216 – 195 (65 * 3) = -9 (MOD 65) = 54 (MOD 65) = M
Plain = 54

Further Explanation:
Euler's was used when possible although this was particularly difficult for the decryption of b)

Exercise 3:
Answer:
a) In order to decrypt the message the private key d is needed, this can not be initially found as it's derivation uses extended Euclidean which needs the value of Phi(N). This value is not obtainable in a realistic time frame due to the usual length of N being too long. If N is particularly small it would be trivial to find these prime factors whoeve I have worked under the assumption that this is not the case. Instead, a common modulus attack can be implimented, this involves Eve taking the two ciphertexts of Alice and Bob, finding the GCD of the two exponents eA and eB (Which is only because N is the same) to receive two

outputs exponents, x and y. Then cA is raised to x before being timesed with cB to the power of y. This leads to the equation:
((cA)^x) * ((cB)^y) == ((MeA)^x) * ((MeB)^y)
and hence by simplifying this equation we output
M^(eA*x + eB * y), when using the extended Euclidean we know
eA*x + eB * y = 1 so M can be found by inputting previously achieved variables

b)
An implimentation in python was completed instead of using cocalc but you may find the testing I did at:
https://cocalc.com/projects/e45c742d-d44b-4fb2-8f47-de9ba8c65f76/files/Part%202.sagews?session=default
I have attempted to add Hasan as a collaborator, please let me know if anything else is needed but the code below should be sufficient. The steps were used from part a) but the only difference is that the negative value for x meant that an inversion was needed, attempts were made to automate this depending on whether x or y was negative but issues occurred so it was completed manually.
M =
1241284880241822245031350316995316509613760031105566488758549213410379930768454638388105627962603033792604848852231910514447243725489419608385858883735979314364859728491299260062917200230440025398527707702850295624884577127897107255287191960914532074227602971007475556486214187806225594131456395882122963770810290613577965251307069511057382696768233468670233045728589973715388024931195704831783334623721062756711756350120344089464621935361024119741178566435368539009948622626793857657155673701127355633007181134673038573037510656910655350038503632683840012771185561459530291941543524137687560199607394816363851431902

Code Implimented (Python):

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

def extendedeuc(a,b):
    r = 0
    q = 0
    lamda11 = 1
    lamda22 = 1
    lamda12 = 0
    lamda21 = 0
    t21 = 0
    t22 = 0
    while b != 0:
        q = int(a / b)
        r = a % b
        a = b
        b = r
        t21 = lamda21
```

```
        t22 = lamda22
        lamda21 = lamda11 - q * lamda21
        lamda22 = lamda12 - q * lamda22
        lamda11 = t21
        lamda12 = t22
    #print (abs(a))
    #print (lamda11)
    #print (lamda12)
    return lamda11, lamda12

def inverse(c, inp, N):
    #print("")
    Alph, Beta = extendedeuc(c, N)
    #print(Alph, "\n")
    inv = Alph % N #Ensures value is low enough to use
    #print(inv)
    #print("\n")
    return (pow(inv, -inp, N))
```

eA =
20645027243220656196109208017037703324445445782163780010862935315414373741837628898839523505123031230795419009232362337631616646221297594562302825854134464078677385857220113598066145202320709652818520766183826127146796762524535662354344777399370862388842166654888842132820363214479571428652601988484224495946375721070361900809019329649759575762397833605938043186770042654743067422241477268365617031178172027178755693566409032818917422094610812754447975895865178682453225583191912212367134587486285799782155109152944427348690163315056825740636659649124179137271843769782829849167189203185398850604993866726625112631363

eB =
345876628472173658667827209598382151105659756356047916616789608178750837186397400300720629651340906876883086914495035331197886605204006436370260001522881159036935864943407229927853909159622840262574082310844414506912374854901914383942533751831444966354228432405145711811352973922639155549967535687324229764372344162085165004285404573177532292355481437060424009838888577080364530392886743183323043816768429349194326761737705660684396204279082233532281958218079492085884478114686098035666343747253331397342623111000982654071759143224287261895337445430397635372322031624510827582994475842436489318181834008077000862716

cA =
599595716443335281998311836057824471423872949240946858680326885165189798248960569757894506358411805849585756670489287378663084704040194999917555981780947347168038953586881042265439088817588061113649282685562685793490922993153055756402440331270258091628957614506114564171089877371669425927413409680720072212671386711718210703493452595971294376452211824154369667228880216368587635945779792347

57925588810668236254574306844174551317112926430234326274195656468
47316989197312413209873196313791487769645026827092218659930429999
11089876791270600102273290388573803610119258785910206047147140049
692741145838272637793573255535
cB =
27708350701504395723615334583238533233311357698898066889526477231
43090100332051468165048542378394448963218541675902047035925067679
29654565693680408876798010395119434074169665510362940394605270654
78936994709171180741516425352827457689751389164261587279622371198
90937352175934340987683034250582715361270008618676286594711314192
18223846481995104293645347796671059631662868723095613383175651057
51299663036243194097923431300592716116535610669280575581993549002
77398160189613164209414757242894166135226111274887275728717755076
38956680302811665295641567387104455860654874353152516271890273421
3942283806624617372725072274578
N =
29691377434483553272708760801010543385405032598789596302501439956
27090090022850067465952425419364381562376747694180021046125245916
81560095590234970477860635176844623820339514255291065927829552481
16268780158586302830558010667680741060756267514670857795628988590
72146739523803688429510424565270597342059874774779917771627069956
14863228176191966838047898473794033307901871686714676364714597531
34984639721096050224406379909319764780531614465362288835895123534
52610817258565587291176075341583042418338206367482892585691607496
65442579435981414832985783135677976190170601699003296209287398379
1585459395968897828575512138547

```
(x,y) = extendedeuc(eA, eB)
#print(x, "\n", y, "\n")
#print("\n")
#cAx = pow(cA,x, N) # Need to inverse as x is negative
cBy = pow(cB,y, N)
#if x < 0:
cAx = inverse(cA,x,N)
#else:
    #cBy = inverse(cB,y,N)

M = cAx * cBy
M = M % N
print(M)
```

Further Explanation:
Euler’s was used when