Summative Assignment - Cryptography

Exercise 1:
Answer:
p = 23
a)

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
5^i MOD 23	1	5	2	10	4	20	8	17	16	11	9	22	18	21	13	19	3	15	6	7	12	14	1

17 * 5 = 50 + 35 = 85 MOD 23 = 4 + 12 = 16
80 = 6 + 5 = 11
55 = 4 + 5 = 9
110 = 8 + 10 = 18
90 = 6 + 15 = 21
105 = 8 + 5 = 13
65 = 4 + 15 = 19
95 = 6 + 20 = 3
70 = 4 + 20 = 1

b) In space Z^*_{23}:
$\log_5 11 = 9$
$\log_5 20 = 5$

Code Implemented: None
Further Explanation:
A technique for question a) was to times the previous number by 5 and then reduce that number by the modulus rather than try to do modulus three to ever higher numbers (25, 125, 625 etc). So for example the answer to 5^2 is 25 mod 23 which gives 2, 5^3 is 125 mod 23 which gives 10 as the answer, the same as 2 (the previous answer) * 5. This kept most of the calculations low enough to calculate in my head and hence not much working.

As shown in lectures, the answers to part b) can be found by using part a) as a lookup table.

Exercise 2:
Answer:
a)
Values used for encryption:
PK = (p,q,g,h1,h2)
SK= (x, y)
0<=z < q
$c_1 = g^z \bmod P$
$c_2 = M \cdot h_1^{-z} \cdot h_2^z \bmod p$

$h_1 = g^x$

$h_2 = g^y$

Finding M which is DEC(SK,Enc(PK, M))

$c_2 = M \cdot (g^x)^{-z} \cdot (g^y)^z$

$c_2 \cdot (g^x)^z \cdot (g^y)^{-z} = M \cdot (g^x)^{-z} \cdot (g^y)^z \cdot (g^x)^z \cdot (g^y)^{-z}$

To cancel the powers on the right hand side

$c_2 \cdot (g^x)^z \cdot (g^y)^{-z} = M \cdot \underline{(g^x)^{-z} \cdot (g^y)^z \cdot (g^x)^z \cdot (g^y)^{-z}}$

So

$M = c_2 \cdot (g^x)^z \cdot (g^y)^{-z}$

or

$M = c_2 \cdot g^{xz} \cdot g^{-yz}$

Using c_1 we can remove g^z

$M = c_2 \cdot g^x \cdot g^{-y}$

leaving $M = c_2 \cdot c_1^{x-y}$

$M = c_2 \cdot c_1^{x-y}$

b)

C = (3, 10)

$c_1 = 3$

$c_2 = 10$

x = 9

y = 8

g = 6

$M = 10 \cdot 3^{9-8}$

$M = 10 \cdot 3^1 = 30 \bmod 23$

$M = 7 \bmod 23$

Code Implemented (Python):

```
#This code does not relate to the answer above, please read
#Further Explanation for more details
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
```

```
    return lamda11, lamda12


M = 7
N = 23
g = 6
x = 9
y = 8
q = 11
p = 23
outc1 = 3
outc2 = 10

for z in range(0, q):
   h1 = (g**x)%p
   h2 = (g**y)%p
   c1 = (g**z)%p
   inp = (h1**z)%p
   (alpha, beta) = extendedeuc(p,inp)
   c2 = (M * beta * (h2**z))%p
   if (outc1 == c1) or (outc2 == c2):
      print("c1:", c1)
      print("c2:", c2)
      print("z:", z)
```

Further Explanation:
I was interested in discovering if you could calculate z once the attacker has M and C, I thought this might have been useful information for if the victim had a predictable z. I attempted a brute force method (which wouldn't work with higher values) and found that z was 7 in this instance. This is interesting as the z is the same as M.

Exercise 3:
Answer:
a)

M = Plaintext
x = Random = Private key
$y = g^x \bmod p$
(p, q, g, y, H) = Public key
0 <= r < q
$C = (g^r , H(y^r) \oplus M)$
$c_1 = g^r$
$c_2 = H(y^r) \oplus M$
(Dec(SK, Enc(PK, M)) = M for KG() = (PK, SK)).
Given C, Public Key and Private Key and hash function for decrypting

Due to the nature of XOR, if $c_2 = H(y^r) \oplus M$, then
$M = H(y^r) \oplus c_2$ and $H(y^r) = M \oplus c_2$
Because the Hash function is accessible and c_2 has already been given, the goal is to find y^r.
$y^{-x} = (g^x)^{-x} \bmod p$
$y^{-X} = g$
$c_1 = (y^{-X})^r = y^{-xr}$
$(c_1)^X = (y^{-xr})^x$ so
$y^r = (c_1)^X$
The first part of the ciphertext is raised to the power of the private key, this can then be hashed and XOR'd with the second part of the message to get the original message, this can be displayed as:
$M = H((c_1)^X) \oplus c_2$

b)
The steps for a chosen-ciphertext attack (CCA attack) to break this encryption are as follows:
We know that:
ENC: $C = (c_1,c_2) = (g^r , H(y^r) \oplus M)$
DEC: $M = H(y^r) \oplus c_2 = (H((c_1)^X) \oplus c_2)$
Initially, two plaintext messages are given to the challenger and the challenger returns one as a ciphertext
$ENC(M\alpha) = (c\alpha_1, c\alpha_2)$
$ENC(M\beta) = (c\beta_1,c\beta_2)$
We are returned $(c\gamma 1,c\gamma_2)$ which can be the encryption of either Mα or Mβ
We again will use the nature of XOR to find both possible Hashes for both our previous Plaintexts
$M \oplus c_2 = H((c_1)^x)$
$M\alpha \oplus c\gamma_2 = H\alpha$
$M\beta \oplus c\gamma_2 = H\beta$
We then create a third plaintext Mθ to use as a contant to then send through the decrypter two ciphertexts:
$(H\alpha \oplus M\theta)$ {0}
$(H\beta \oplus M\theta)$ {1}
One of the outputs from these two decryptions will be the same as one of the input plaintexts, we can then return 0 or 1 depending on the correct answer, winning the game with 100% accuracy.

Code Implemented: None
Further Explanation:
To answer this question several assumptions are made, they include:

- All ciphertexts given into the system are padded until the same length
- The size of all variables is high enough that brute forcing isn’t feasible in a reasonable time frame
- The hashing algorithm does not produce collisions easily, if this is the case then we may not be able to effectively differentiate texts
- The value entered into the DEC can not be equal to the ciphertext given by the challenger, the use of c1 or c2, the seperate sections of the challenger ciphertext, is allowed
- You may send more than one ciphertext throught the decrypter (This example can be done with one however if necessary)