CryptoHomework5

February 25, 2020

[ ]:
```
'''
CS 764 Blockchains and Cryptocurrencies
Module 5, Homework 5
February 18, 2020
Joseph S Cannella
'''
```

[39]:
```
################################
# Q1. Blind Digital Signatures #
################################

# (i) Given p=5,q=11 determine public key (e,n) and private key (d,n) Where e=7
p = 5
q = 11
e = 7# Public Key (e,n)
n = p*q# base (55)
Phi_n = (p-1)*(q-1)
print('n = %d' %n)
print('Phi(n) = %d' % Phi_n)

# Note: d == e^-1(mod(Phi_n)), or e*d = 1 mod Phi_n, 7*d = 1 mod 40
# Define a helper function for obtaining modular inverse
# Refference: egcd() and modinv() were obtained from following site
# https://stackoverflow.com/questions/4798654/
↪modular-multiplicative-inverse-function-in-python
def egcd(a, b):
    if a == 0:
        return (b, 0, 1)
    else:
        g, y, x = egcd(b % a, a)
        return (g, x - (b // a) * y, y)

def modinv(a, m):
    g, x, y = egcd(a, m)
    if g != 1:
        raise Exception('modular inverse does not exist')
    else:
```

```
        return x % m
# Determine the Value of the Private Key
d = modinv(e, Phi_n)
print('d = %d' % d)
```

```
n = 55
Phi(n) = 40
d = 23
```

[40]:
```
# (ii) Given: b=6 determine blinding factor using derived public key
b = 6 # randomly chosen number used to blind the message
BF = pow(b, e) % n
print('Blinding Factor = %d' % BF)
```

```
Blinding Factor = 41
```

[41]:
```
# (iii) Given: m=11, determine blinded message
m = 11# Plain Text Message
BM = m*BF % n# Blinded Message
print('Blinded Message = %d' % BM)
```

```
Blinded Message = 11
```

[42]:
```
# (iv) Determine signature on blinded message using private key
BM_signed = pow(BM, d) % n# Bob's signature on blinded message
print('Signed Blinded Message = %d' % BM_signed)
```

```
Signed Blinded Message = 11
```

[43]:
```
# (v) Unblind the signature on the blinded message to obtain signature on␣
 ↪message
SignedMessage = BM_signed*modinv(b, n) % n# Original Message signed by Bob
print('Signed Message = %d' % SignedMessage)
```

```
Signed Message = 11
```

[44]:
```
# (vi) Verify blind signature matches signature on m using (d,n)
VerifiedSignature = pow(m, d) % n
print('Verified Signature = %d' % VerifiedSignature)
```

```
Verified Signature = 11
```

[13]:
```
###############################
# Q2, Bit-commitment protocol #
###############################
import hashlib
# Secret Nonces for each person (Alice, Bob, Carol, David)
NA = 2
```

```
NB = 7
NC = 3
ND = 5

# Secret Predicted Winner John:0, Jane:1
PWA = 0
PWB = 0
PWC = 1
PWD = 1

# Public Hashes
HashA = hashlib.md5()
HashA.update((str(NA) + str(PWA)).encode('utf-8'))
HashB = hashlib.md5()
HashB.update((str(NB) + str(PWB)).encode('utf-8'))
HashC = hashlib.md5()
HashC.update((str(NC) + str(PWC)).encode('utf-8'))
HashD = hashlib.md5()
HashD.update((str(ND) + str(PWD)).encode('utf-8'))

# Review Hashes
print('HashA: ', HashA.digest())
print('HashB: ', HashB.digest())
print('HashC: ', HashC.digest())
print('HashD: ', HashD.digest())
```

```
HashA:  b'\x98\xf17\x08!\x01\x94\xc4uh{\xe6\x10j;\x84'
HashB:  b'|\xbb\xc4\t\xec\x99\x0f\x19\xc7\x8cu\xbd\x1e\x06\xf2\x15'
HashC:  b'\xc1jS \xfaGU0\xd9X<4\xfd5n\xf5'
HashD:  b"(8\x02:w\x8d\xfa\xec\xdc!'\x08\xf7!\xb7\x88"
```

[ ]:

[33]:

```
###########################
# Q3. Zero-Knoledge proofs #
###########################

# Remote client C proves to server S it knows password
# wihtout actually sending it across internet

# (i) C Sends username to S
username = 'Alice'

# (ii) S Verifies username and sends nonce N to C
if(username == 'Alice'):
    Nonce = 2357
```

```python
# (iii) C Computes hash H=MD5(N||P||N) and sends H to S
HashZero = hashlib.md5()
HashZero.update((str(Nonce) + "SecretPassword123" + str(Nonce)).encode())
HashZero = HashZero.digest()
print('HashZero: ', HashZero)
```

```
HashZero:  b'\x15; C\x10~imA\xd1S\x1a\xfd;\x7f\x95'
```

[34]:
```python
# (iv) S independently computes G = MD5(N||P||N) and compares
HashG = hashlib.md5()
HashG.update((str(Nonce) + "SecretPassword123" + str(Nonce)).encode())
HashG = HashG.digest()
print('HashG: ', HashG)
```

```
HashG:  b'\x15; C\x10~imA\xd1S\x1a\xfd;\x7f\x95'
```

[35]:
```python
if HashZero == HashG:
    print("Verified!")
else:
    print("Invalid!")
```

```
Verified!
```

[3]:
```python
#####################################
# Q4. Discrete Zero Knoledge Proofs #
#####################################

# Given: Alice and Bob know public values p=17 and A=5
# Alice also knows x=10
# Show: How Alice can prrove she knows x w/o revealing it to Bob

# Public Variables
p = 17
A = 5

# Alice's Private Variables
x = 10


# 1.) A, B, and p are public

# 2.) Alice  Computes A^x mod p
# Note: A^x = B mod p
B = pow(A, x) % p
print('B = %d' % B)

# Helper Function
```

```python
from random import randint

# 3.) Alice chooses random number r < p and sends q = A^r mod p to Bob
r = randint(0, p-1)
print('r = %d' %r)
q = pow(A, r) % p
print('q = %d' % q)

# 4.) Bob sends random bit i
i = randint(0, 1)
print('i = %d' % i)

# 5.) Alice Computes s=(r+i*x) mod p and sends s to Bob
s = (r + i*x) % (p - 1)
print('s = %d' % s)

# 6.) Bob computes C=A^s mod p, which chould be equal to D=q*B^i
# If it doesnt then he knows Alice doesn't know x
C = pow(A, s) % p
print('C = %d' % C)
D = q*pow(B, i) % p
print('D = %d' % D)
if(C == D):
    print('Alice knows the value of x')
else:
    print('Alice does NOT know the value of x')
```

```
B = 9
r = 15
q = 7
i = 1
s = 9
C = 12
D = 12
Alice knows the value of x
```

[ ]:

```python

```