Part A

Check that the system parameters *p*, *q*, and *g* satisfy the three criteria in Table 1. You need to provide the factorization of *p – 1*, and explain your verification process. Explain why we only pick sk_1 as a number less than 224 bits. Compute pk_i, *i* = 1, 2, 3.

A1 - *p*, a prime number lying between 1024 and 2048

```
# binary literal starts from index 2 onwards of bin(n) output

p = 1615850420240242625399113195036680055148205339919365512280505165762970604025264132936922942592721900695647374247690

len(bin(p)[2:])
2048

is_prime(p)
True
```

Figure 1.1 – Verification of A1: p is a prime number lying between 1024 and 2048 bits

A2 - *q*: a 224-bit prime factor of *p* - 1

```
q = 13479974306915323548855049186344013292925286365246579443817723220231

len(bin(q)[2:])
224

(p - 1) % q == 0
True

is_prime(q)
True
```

Figure 1.2 – Verification of A2: q is a 224-bit prime factor of p – 1

A3 - *g*: an element g ∈ GF(*p*) with order *q*

g must satisfy the following criteria:

$$\boldsymbol{I.\, g^{p-1} mod(p) = 1}$$

We can prove this criterion using *Fermat's Little Theorem* which states that:

For any modulus *p* and integer *g* coprime to *p*, one has

$$g^{\varphi(p)} \equiv 1$$

Where $\varphi(p)$ denotes Euler's totient function (which counts the integers from 1 to *p* that are coprime to p). Fermat's little theorem is a special case, because if p is a prime number, then $\varphi(p) = p - 1$. Using Fermat's Little Theorem we get:

$$g^{p-1} mod(p) = 1$$

$$II.\ g^r mod(p) \neq 1\ \forall\ 1 \leq r < p - 1$$

For this criterion, we only need to verify this case holds for all r that are prime factors of $p - 1$ (assuming there are 3 prime factors):

Using SageMath's elliptic curve factorization module (ECM), we are able to perform a factorization of p – 1 using the find_factor method:

```
ecm.find_factor(p-1)

[2,
 807925210120121312699556597518340027574102669959682756140252582881485302012632066468461471296360950347823687123845198939436418633983128063387479637823929232659368983403503873582011652663393397044988113492776924811463632313363009966597537728091347905766198208378615605634161718437279159494424968184640409761402781930816777116412062115800159424553847651448925953211117393936233416181059782564167068942256420063165653546611480647177784653819204704796160494415949171918711837809729492566983643659716312327697754540269919577509638499721929712158912138330944190162806056107354164991070604554758310699571997946300780348 6771]
```

Figure 1.3 – Factorization of p – 1 using the find_factor method from SageMath's ECM module

We know p is an odd number, so $p - 1$ is an even number and therefore its first prime factor must be 2. Then we check to see whether or not the other factor, u, returned by this function is a prime number:

```
u = 8079252101201213126995565975183400275741026699596827561402525828814853020126320664684614712963609503478236871238451

is_prime(u)

False
```

Figure 1.4 – Negative primality test of the factor u returned from the find_factor method using p - 1

Since u is not a prime factor, we recursively divide $p - 1$ by q *(a prime factor of p - 1)* and verify if the second value returned by the find_factor function is a prime number (since the first value returned is always 2). Fortunately, only one iteration is required before determining the other prime factor of $p - 1$, v:

```
ecm.find_factor((p-1)/q)

[2,
 599352188457547639693740171522680835865322184075286620256386716439921029334782998562008313995103840817116953210359643511447372101221464995653177706653918911634015555633001801773590292023083604552613918655194669929368962688659673544061885990566694774938089095597940505443430714573194211134223784784744939911387314440899638056016474270609853063142638329500429039904897328933858412897374253962878313102199163400319655392723027703101354927814377851954345336880097584669598515815463134218486896858352351187255794957262790967727451704317515673833695348341]

v = 5993521884575476396937401715226808358653221840752866202563867164399210293347829985620083139951038408171169532103596

is_prime(v)

True
```

Figure 1.5 – Positive primality test of the factor v returned from the find_factor method using (p – 1)/q

We now have 3 prime factors for $p - 1$: 2, q, and v and can verify the following criterion:

$$II.\ g^r mod(p) \neq 1$$

```
pow(g, 2, p) == 1
False

pow(g, q, p) == 1
True

pow(g, v, p) == 1
False
```

Figure 1.6 – Verification of second criterion to determine if g is an element of GF(p)

Note that $g^r mod(p) = 1$ for $r = q$. Lagrange's theorem states that the order of any subgroup of a finite group divides the order of the entire group. If g is any number coprime to p then g is in one of these residue classes. Thus, group element g has finite order q, and its powers g, g_2, ..., g_k (mod p) form a subgroup of the group of residue classes, with $g_k = 1$ (mod p). Consequently, Lagrange's theorem states that q must divide $\varphi(p)$.

```
(p-1) % q
0
```

Figure 1.7 – Verification that q divides p - 1

Since q divides $p – 1$, and because there exists no prime factor of $p – 1$ lesser than q which satisfies this condition, we know that q is the order of the element g, which has also been verified as a primitive element of GF(p) given that the other prime factors of $p – 1$ satisfied condition *II*.

Why we pick sk_i as a number less than 224 bits

sk_i is used in two places – computing the corresponding pk_i, and in the signing function. In each location, sk_i is used in an arithmetic operation which incorporates a modulo, meaning that regardless of the key value, the result of the operation is bounded from zero to the divisor minus one. As a result, extending the length of sk_i beyond the requirement does not provide any additional security.

pk_i computation

For $i = 1, 2, 3$:

```
pk_1 = pow(g, sk_1, p)
pk_2 = pow(g, sk_2, p)
pk_3 = pow(g, sk_3, p)
```

Figure 1.8 – Function used to compute pk_i

The above values can be found in the Appendix.

Part B

Sign and validate transactions by DSA (i.e. DSS, the signing equation defined in NIST FIPS 186-4): implement a DSA module which enables the user can sign the transactions and a miner can verify the signed transaction. (Note that since we set all inputs to zero, so $m_i = Tx_i$, i = 1, 2, 3.)

1) **User 1: sign his transaction: $Sig_{sk_1}(m_1) = (r_1, s_1)$ i.e, compute the signature over $m_1, k_1, r_1 = \left(g^{k_1} mod(p)\right) mod(q), k_1{}^{-1} mod(q)$ and $s = k_1{}^{-1}\left(h(m_1)\right) + xrmod(q)$**

In accordance with the NIST FIPS 186-4 specification, the signature generation function was implemented using the following function provided in section 4.6 DSA Signature Generation:

```
# DSA signature function, p, q, g, k, sk are integers, Message are hex strings of even length.
def Sign( p, q, g, k, sk, Message ):
    k_inv = inverse_mod(k, q)
    if k_inv is not None:
        r = pow(g, k, p) % q

        N = len(bin(q)[2:])

        h_m = sha3_224_hex(Message)
        binary_h_m = str(bin(int(h_m, 16)))[2:].zfill(len(h_m[2:] * 4))
        outlen = len(binary_h_m[2:])
        z = binary_h_m[:min(N, outlen)]

        z_10 = int(z, 2)
        s = (k_inv * (z_10 + (sk * r))) % q

        if r != 0 and s != 0:
            return r,s

        raise Exception("either r or s were equal to 0 - generate a new value for k")
    raise Exception('k-1 was not found')
```

The set of input parameters *p*, *q*, *g*, k_1, sk_1, and m_1, and output parameters r_1 and s_1 can be found in the Appendix.

2) **A miner: verify $Sig_{sk_1}(m_1)$, i.e., compute the values of u, v, w, and verify whether v = r.**

In accordance with the NIST FIPS 186-4 specification, the signature verification function was implemented using the following function provided in section 4.7 DSA Signature Verification and Validation:

```
# DSA verification function,  p, q, g, k, pk are integers, Message are hex strings of even length.
def Verify( p, q, g, pk, Message, r, s ):
    M_prime = Message
    r_prime = r
    s_prime = s
    y = pk
    N = len(bin(q)[2:])

    if not (0 < r_prime < q) or not (0 < s_prime < q):
        return False

    w = inverse_mod(s_prime, q)
    # print('w: ' + str(w))
    if w is not None:
        h_m = sha3_224_hex(M_prime)
        binary_h_m = str(bin(int(h_m, 16)))[2:].zfill(len(h_m[2:] * 4))
        outlen = len(binary_h_m[2:])
        z = binary_h_m[:min(N, outlen)]
        z_10 = int(z, 2)

        u1 = (z_10 * w) % q
        # print('u1: ' + str(u1))
        u2 = (r_prime * w) % q
        # print('u2: ' + str(u2))
        v1 = pow(g, u1, p)
        v2 = pow(y, u2, p)
        v = ((v1 * v2) % p) % q
        # print('v: ' + str(v))

        return True if v == r_prime else False
```

The set of input parameters p, q, g, $pk1$, m_1, r, and s can be found in the Appendix. The output produced by the set of input parameters was True. The intermediate variables w, u_1, u_2, and v can be found in the Appendix.

3) **User 2: sign his transaction $m_2 = Tx_2$ using the key pair (sk_2, pk_2), i.e. executing the same steps as user 1.**

Using the function from part 1) The set of input parameters p, q, g, k_2, sk_2, and m_2, and output parameters r_2 and s_2 can be found in the Appendix.

Part C

Proof-of-Work (PW): Implement a module for a miner to compute a PW where SHA3-224 is used as a hash function *h* in *PW₁* and *PW₂* computations.

1) **Find pre-images of *h* such that**

$$\boldsymbol{PW_1 = h\big(h(amt_0)||m_1||nonce_1\big) = 00\dots0 ** \dots *}$$
$$\boldsymbol{PW_2 = h\big(h(m_1)||m_2||nonce_2\big) = 00\dots0 ** \dots *}$$

where * means any value and $nonce_i$, i = 1, 2 are any 128-bit numbers. Here you use k = 32. You should vary a none in order to obtain a SHA3-224 hash value with 32-consecutive leading zeroes. Your results on hash values PW_1 and PW_2 should be represented as hexadecimal numbers.

Both $nonce_1$ and $nonce_2$ (can be found in the Appendix) were generated using the following brute-force procedure:

```
def nonce(message):
  i = 0
  max_val = (2 ** 128) - 1
  while True:
    nonce = '{0:0128b}'.format(i)

    input = message + nonce

    # binary_string_to_hex first converts input to integer - causes loss of leading zeroes in returned hex string
    hex_input = binary_string_to_hex(input)

    # pad hex input with leading zeroes to account for loss; 1114 is the expected length of the hex string
    pad = 1114 - len(hex_input)
    padded_hex_input = ('0' * pad) + hex_input

    output = hex_to_binary_string(sha3_224_hex(padded_hex_input))
    if output[:32] == '{0:032b}'.format(0):
      return nonce
    if i == max_val:
      return None
    i += 1
```

The respective pre-images PW_1 and PW_2 (can be found in the appendix) were generated using the following function:

```
def construct_pre_image(arg1, arg2, nonce):
    return sha3_224_hex(binary_string_to_hex(hex_to_binary_string(arg1) + arg2 + nonce))
```

where $arg_1 = h(amt_0)$ and $arg_2 = m_1$ for PW_1, and $arg_1 = h(m_1)$ and $arg_2 = m_2$ for PW_2.

2) **Determine the average number of trials which you need to get one PW in (1).**

Given that the computed pre-image is 224 bits in length and the first 32 bits are set to zero, the remaining 192 bits can be any combination of zeroes and ones. As a result, there are 2^{192} possible pre-images with 32 leading zeroes out of a total 2^{224} combinations. Therefore, the probability of finding the correct pre-image with 32 leading zeroes is:

$$\frac{2^{192}}{2^{224}} = \frac{1}{2^{32}}$$

Part D

Security analysis: Discuss why the PW can prevent double spending in the Bitcoin network and identify two possible attacks on PW.

Two transactions must be performed to attempt double spending – the second transaction will have the same origin as the first and can therefore be easily identified as an attack. When a miner finds the corresponding pre-image, broadcasts it, and has it verified by other miners, the hash-chain will add a block with the signed transaction and the pre-image. In other words, the transaction data is stored permanently in the blocks. Without a pre-image, a block cannot be added to the pre-existing chain. If this were not the case, an attacker would be able to create another transaction using the bitcoins from previous transactions and there would be no way of verifying whether or not the next transaction was previously processed.

51% attack: In the bitcoin system, any group of miners who control greater than 50% of the computing power of the network are in possession of majority control. Consequently, they are able to interrupt the addition of new blocks by preventing other miners from completing them.

MITM: an attacker can intercept a miner's broadcasted pre-image and replace it with an invalid pre-image, keeping the valid pre-image for themselves to broadcast instead.

Appendix

p=16158504202402426253991131950366800551482053399193655122805051657629706040252641329369229425927219006956473742476903978788728372679662561267749592756478584653187379668070077471640233053267867940899762269855538496229272646267260199331950754561826958115323964167572312112683234368745583189888499363692808195228055638616335542328241242316003188491076953028978519064222347878724668323621195651283341378845128401263313070932229612943555693076384094095923209888318983438374236756194589851339672873194326246553955090805398391550192769994438594243178242766618883803256121122147083299821412091095166213991439958926015606973543

q=13479974306915323548855049186344013292925286365246579443817723220231

g=989166310174906059611052564880044231226204762170000871033229080335441973441523940037409297250576036855503397888372709087879878652786910610212556867451508776729606489881356330549169747474399916453864516259348034061458342027269766945943995605795775664653137969485217890077966731174553543597150973233536157598924038645446910353512441488171918287556367865699357854285249284142568915079933750257270947667792192723621634761458070065748588907955333315440434095504696037685941392628366404344728480845324408489345349308782555446303365930909965625721154544418491662738796491732039598162639642305389549083822675597763407558360

k_1=7636180595255512892709086526638263042018821609914680860810683215268

sk_1=4001206756878977378588887638958103884884701595184090082238631140177

m_1=0110100001111100011100101000000111001000101011111001001111100010000010100101110010000101101010101100111001011010001111110000111001000111011110010101101000000000001101101101100111001100000101010011111011100000110111100011001000000111010101100011111010111001001111010110011101110010110011000111111110100111000100010101001001111011110100100100111010111000010010010010110011101000000101101100110000110010000001011101001110111100000111101000011100010110011000000001010000001001111011010101010111100100101000111011111011101111011000111100001000100111010011001000000000100111001111110101000000111001111000011010100011001011011111010001000001000101001101000111100010110101011001111000011110010111010001100111010110011010001010111101101010111101111111111110101101010011110101010101010000011100011010000101001001010100110001011000000011111000000111100100111111011000001010010100001101001100110100010001100001111100010010011010110110110000101101010101110011000010100001101000011101101110100101101100010100111111001000111101001011001101100010110100100101011010000110110100100110001010100011000010001100000101011101010101100010010001011110001100001110100010011110101011110000000101110100110011001001001101010011101100110111001111101001111110111110011110111101110001111011110011101011101000110001001001100101111110010011010001101111110101000111011101111010100000111100000001111010010001011101001110100000101100000011101010100111000100011010110111101100111011000000011110111000101011011101010001111100100100001010001000001110010100101000011110001000010101110011011110110010011010001001001111110000010011111011111100001000110001011101111101000101001100111110100010000111000000110101111000111110100011110101001011000110011111001011001101101111000110011000100000111001000011001111111111111110111010111011010101001100111011001011100100111100011100101001010100000000000000101010001011000011011111011101110100110110010011100111110100101110110001111011101011011101001100011100001111011010101011100111111010111001001111001011101011011110110011111111001101010100011110111011100110001101011100101010000110010111011000001010111011101110000001111110001101011010101100100110111111010010000101000011110001001101100010111000011100111011110001010100100100111110101101001010010100110111011110000101010111110010001111100110011000010000111111101100001100111001100010101111011101000100011001001110111111011010010010110010100000000000110110011110100011111100000001101101001000110000111010110101010101001011010001111111111110011011000100011100101000110010010000000001000101011111111000011000101010111001101000010001101001001011111100110000000101000000101010100001100010111111110001010010010111010111000100010001101110110111101000000000111001010000011011101101010010101101100000111001010001100101111111101001101010111101000101000001101001000001110010110100110000110100010101000111111100111010001001100100111101001100111000000111100101110100000111011011001011001101000100001000101010000011101110000001011110000000110111001010100111101000010111001111110001001011001101101000110011000010110001111011010100111111001011010000101101010010000111010101000010111101000001101111111110001110001111001110010000000001010101110111011001010000000100011001110000001110010011101110011010110101110000011101001110100010110110011011000011101100000100111110001001110110010011111001111110100110010111110011111100100110010101101001110100001000100010011110000110110001001110010011000010011000110110010111101011001100100000110000110001101011101101101000111011100011000111001000100000101101111001001101101100110010100010000100111110000111111011011010101111001011100011011110011010100111111110111011001100011010111110010010001001001010100010111010010011001101101111001110101010001000110111100001000100000000011111000100110100001001101011001000110000000110011011010000011001100001000011011111000110110001101111011001001100001111111110111111100101111000111010011011010110010111111100011100100100100000011001000110111111011000100010011011111000110000000100001101101100110001011000000101111000010100111101000110101000110111010110011011111000000101001111001111010011110111010011101010010001111110101000000000100

pk_1=13190150963760889647094634684573736595322369628268521716277176722504680205306666957427063669382047355005579753986408363017207529087738312031354846179411020392929821489131982024455275601635702544446679012547008732424008115598107948911123181902989407565493889955188285972381676176352955758942149787306114718970006328435881616077569969216032235994360740773429218909181563508361363587117820083370967410980903079772729089192202299692711592106070919451037142106787287918678319935616746224007115660659895996235128698926340666694953849011519321142057025235977157794326767978009471826246937284652336421864460645512691392114275

r_1=5102491627416874343359069298372509963208458310973379563071871951255

s_1=2231571007439116961204453230930081120395191936038941107639012237189 w=79199880376496508413748633491484728392296210608377496018119457636 81

u1=1932139136555447985697411803330180555739694494026775736635882058118

u2=1211982009756804633432366613363588068075205947705932733709010798536 3

v=5102491627416874343359069298372509963208458310973379563071871951255

k2=1126289620759427337161193756796339715982474853764963010968386403961

sk2=5779569239846031148528787672249249942175134704083930000648821513800

m2=0101110010101000011001011101100000101011101110111000000111111000110101101010110010011011111010010000101000011110 0
0100110110001011100001110011101111000101010010010011111010110100101001010011011101111000010101011111001000111110011 00
110000100001111111011000011001110011000101011110111010001000110010011101111101101001001011001010000000000110110011110
100011111100000011011010010001100001110101101010101010010110100011111111111001101100010001110010100011001001000000001
000101011111111000011000101010111001101000010001101001001011111100110000000101000001010101000011000101111111000101001 00
1011101011100010001000110111011011110100000000011100101000001101110110101001010110110000011100101000110010111111110 10
011010101111010001010000011010010000011100101101001100001101000101010001111111001110100010011001001111010011001110000
011110011011101000001110110110010110011010001000010001010100000111011100000101111000000110111001010100111101000010111
001111111000100101100110110100011001100001011000111110110101001111100101101000010110101001000011101010100001011110100 0
001101111111100011100011110011100100000000010101011101110110010100000100011001110000011100100111011100110101101011100
001110100111010001011011001101100001110110000010011111000100111011001001111100111110100110010111111001111110010011001 0
101101001110100001000100010011110000011011000100111001001100001001100011011001011110101100110010000011000011000110101 1
101101101000111011100011000111001000100001011011110010011011011001100101000100001001111100001111110110110101010111100 10
111000110111110011010100111111111011101100110001101011111001001000100100101010001011101001001100110110111100111010101 00
010001101111100001000100000000001111100010011010000100110101100100011000000110011011010000011001100001000011011111000 11
011000110111101100100110000111111111101111110010111100011101001101101011001011111110001110010010010000001100100011011
111101100010001001101111100011000000100001101101100110001011000000101111000010100111101000110101000110111010110011011 1
110000010100111100111101001111011101001110101001000111111010100001101101101100011111001101111110000100001001000000001
100010110111011001110111001110011001110111110011101001000011010110011101010101000101101101000100000110011101011010111
101111111101010110111101100111101011000000000000011010111010001111010001111111010000000101101100011011101000000011001 0
000010001100111110000011000111000111000110111010011010001001111011000010000001000000010011000011101110000110101000100 0
100001111000110000001101110001110111000000101011000110101101100110010101001110001111000111000000100110011010001111001 010
000001101011001011100110101010011101100101011010100101010011110111010010110111111111100111111010000000111011110000001 0
010111010111100010110100101010001010100010101010000011100111011001001110000011010100000000100101100001111000000001111 00
010011110011011001010000000101111011110101001011101000101001100100011101101111111111011001010011001111100011101111010 11
011100011010100110001101001111000101101110000111010100001001110000000010010100111011000101100111011000110010010110001 0
010011111000111011110100101000010011011011000000110100101000001001110100001100001010010100110001001000011100100001010
010001100101011000110000100011101110010011101011100010001011110101110001011010100100000010011110110100000000101011001 0
001001010001000111010101000110100101111000110111111000010111010110000100100000010111001101011001110111100110010101111 11
010111110111101011111000100010000101010111111001011011010100010001010011000101100110001000100011001100011101001100111
000000011111010101011110100101010101111111010001010011110101110100011000101111001011101100011001001100100001001000011 1
011100111101010100100111011010111101011011001100011001010110101110000110010011110110101111100111001010000111011011 00
100111010100110110110100011110101001110001100101111111111011111111010010100111111001101010100111100000001101010110 1100
101000001101000010011110000010011111011010010100011111010011010111111111111010010110001111000110011110101110001111101 10
101001100111100101111010010011010111010101100101101000011010111100101001011111101011010101001101100000110111111100011 1
1101000000011

r2=8516160244583270803284104545316545677581615323124894373363462902764

s2=9364283268950232159179379840816945956738425736262891205114832593141

nonce1=0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000010011111101000
011010011101100010

nonce2=0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000011001110101001
101000101101101101

PW1=0000000085216dc453c650a7a60aed9b6ee5d1ef152a3905b0308a77

PW2=00000000fba516c7df21a357340ba3107ccc71022479adcaaf21e2aa

pk2=116969641216752294746525086864650022239105494689273511558184700518925326094443963570940370686541826189443217594 66
7947377981298825687586352944378225295819488843687939082175771643713407614937489732020903860897841173484448963383330 45
6238272637763515306774894235109904016146754986513688065397008926656307948279700745395025196369856357034239048462678 73
2686380270005128690858963860075892888747919411040242089051794715656968608985157420507496270501604020911470504318619 07
8604108531944065780922866814848153114943920506250259473470916000112353792530966684068678513729288769661498167076188 21
6780607373761507954761915722422394 44

pk_3=69238631100571491066665417485592019917199099742539341201263231895780674491870379970783473833266001044401042034856
80439119500162846548346285729910618221214545837951706394484426133301742347021603115532900270639410225516197623 6246938
997529442889556057234561378726825116615609389426339357733824659845686727507756443831545762362887505137591329563670395
004532108723734106376953903841478881633562942554761025134850977453613577924090641747456023771908740547671011951813695
009218059625385445239785403796170915414450712050972677065846116720557081721159899303888744110775576320091960341713585
59384442134475394473727405232879866