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 = 16158504202402426253991131950366800551482053399193655122805051657629706040252641329369229425927219006956473742476900

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:

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

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

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,
 807925210120121312699556597518340027574102669959682756140252582881485302012632066468461471296360950347823687123845198939436418633983128063387479637823929232659368983403503873582011652663393397044988113492776924811463632313363009966597537728091347905766198208378615605634161718437279159494424968184640409761402781930816777116412062115800159424553847651448925953211117393936233416181059782564167068942256420063165653546611480647177784653819204704796160494415949171918711837809729492566983643659716312327697754540269919577509638499721929712158912138330944190162806056107354164991070604554758310699571997946300780348677 1]
```

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 = 807925210120121312699556597518340027574102669959682756140252582881485302012632066468461471296360950347823687123845
```

```
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 = 599352188457547639693740171522680835865322184075286620256386716439921029334782998562008313995103840817116953210359
```

```
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:

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

```
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 $s = {k_1}^{-1}\left(h(m_1)\right) + xr mod(q)$**

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 PW_1 and PW_2 computations.

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

$$PW_1 = h\big(h(amt_0)||m_1||nonce_1\big) = 00\ldots0**\ldots*$$
$$PW_2 = h\big(h(m_1)||m_2||nonce_2\big) = 00\ldots0**\ldots*$$

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=161585042024024262539911319503668005514820533991936551228050516576297060402526413293692294259272190069564737424769
039787887283726796625612677495927564785846531873796680700774716402330532678679408997622698555384962292726462672601993
319507545618269581153239641675723121126832343687455831898884993636928081952280556386163355423282412423160031884910769
530289785190642223478787246683236211956512833413788451284012633130709322296129435556930763840940959232098883189834383
742367561945898513396728731943262465539550908053983915501927699944385942431782427666188838032561211221470832998214120
91095166213991439958926015606973543

q=13479974306915323548855049186344013292925286365246579443817723220231

g=989166310174906059611052564880044231226204762170000871033229080335441973441523940037409297250576036855503397888372
709087879878652786910610212556867451508776729606489881356330549169747474399916453864516259348034061458342027269766945
943995605795777566465313796948521789007796673117455354359715097323353615759892403864544691035351244148817191828755636
786569935785428524928414256891507993375025727094766779219272362163476145807006574858890795533331544043409550469603768
594139262836640434472848084532440848934534930878255544630336593090996562572115454441849166273879649173203959816263964
2305389549083822675597763407558360

k_1=7636180595255512892709086526638263042018821609914680860810683215268

sk_1=4001206756878977378588887638958103884884701595184090082238631140177

m_1=01101000011111000111001010000001110010001010111110010011111000100000101001011100100001011010101011001110010110100
011111000011100100011101111001010110100000000000110110110110011100110000010101001111101110000011011110001100100000011
101010110001111101011100100111101011001110111001011001100011111110100111000100010101001001111011110100100100111010111
0000100100100101100111010000010110110011000011001000001011101001110111100000111101000011100010110011000000001010000001
001111011010101011110010010100011101111101110111101100011110000100010011101001100100000000100111001111110101000000111
001111000011010100011001011011111010001000001000101001101000111100010110101011001111000011110010111010001100111010110
011010001010111101101010111101111111111110101101010011110101010101010000011100011010000101001001010100110001011000000
0111100000001111001001111110110000010100101000011010011001101000100011000011111000100100110101101101100001011010101011
100110000101000011010000111011011101001011011000101001111110010001111010010110011011000101101001001010110100001101101
0010011000101010001100001000110000001010111010101011000100100010111100011000011101000100111101010111100000001011101001
1001100100100110101001110110011011100111110100111111011111100111101111011100011110111100111010111010001100010010011001
0111111001001101000110111111010100011101110111101010000011110000001111010010001011101001110100000101100000001110101010
0111000100011010110111101100111011000000001111011100010101101110101000111110010010000101000100000111001010010100001111
0001000010101110011011110110010011010001001001111110000010011111011111110000100011000101110111110100010100110011111010
0010000111000000011010111100011111010001111010100101100011001111100101100110110111100011001100010000011100100001100111
1111111111101110101110110101010011001110110010111001001111000111001010010101000000000000001010100010110000110111110111
01110100110110010011100111101001011101100011110111010110111010011000111000011110110101010111001111110101110010011110
0101110101101111011001111111110011010101000111101110111001100011010111001010100001100101110110000010101110111011100000

01111110001101011010101100100110111110100100001010000111100010011011000101110000111001110111100010101001001001111010
11010010100101001101110111100001010101111100100011111001100110000100001111111011000011001110011000101011110111010001 0
00110010011101111101101001001011001010000000000110110011110100011111100000011011010010001100001110101101010101010100101
10100011111111111001101100010001110010100011001001000000001000101011111110000110001010101110011010000100011010010010 1
1111100110000001010000010101010000110001011111110001010010010111010111000100010001101110110111101000000000111001010 00
00110111011010100101011011000001110010100011001011111111010011010101111010001010000011010010000011100101101001100001 1
01000101010001111111001110100010011001001111010011001110000011110011011101000001110110110010110011010001000010001010 1
00000111011100000101111000000110111001010100111101000010111001111110001001011001101101000110011000010110001111101101 0
10011111001011010000101101010010000111010101000010111101000001101111111000111000111100111001000000000101010111011101
10010100000010001100111000001110010011101110011010110101110000111010011101000101101100110110000111011000000100111110001
00111011001001111100111110100110010111110011111100100110010101101001110100001000100010011110000110110001001110010011 0
00010011000110110010111101011001100100000110000110001101011101101101000111011100011000111001000100001011011110010011 0
11011001100101000100001001111110000111111011011010101111001011100011011110011010100111111110111011001100011010111110 01
00100010010010101000101110100100110011011011110011101010100010001101111000010001000000000111110001001101000010011010 1
10010001100000011001101101000000110011000010000110111110001101100011011110110010011000011111111101111111001011110001 11
01001101101011001011111110001110010010010000000110010001101111111011000100010011011111100011000000100001101101100110001 0
11000001011110000101001111010001101010001101110101100110111110000010100111100111101001111011101001110101001000111111 0
1010000000100

pk_1=1319015096376088964709463468457373659532236962826852171627717672250468020530666695742706366938204735500557975398 6
40836301720752908773831203135484617941102039292982148913198202445527560163570254444667901254700873242400811559810794 8
91112318190298940756549388995518828597238167617635295575894214978730611471897000632843588161607756996921603223599436 0
74077342921890918156350836136358711782008337096741098090307977272908919220229969271159210607091945103714210678728791 8
67831993561674622400711566065989599623512869892634066669495384901151932114205702523597715779432676797800947182624693 7
2846523364218644606455126913 92114275

r_1=5102491627416874343359069298372509963208458310973379563071871951255

s_1=22315710074391169612044532309300811203951919360389411076390122 37189

w=79199880376496508413748633491484728392296210608377496018119457 63681

u_1=1932139136555447985697411803330180555739694494026775736635882058118

u_2=1211982009756804633432366613363588068075205947705932733709010798536 3

v=5102491627416874343359069298372509963208458310973379563071871951255

k_2=112628962075942733716119375679633971598247485376496301096838640396 1

sk_2=5779569239846031148528787672249249942175134704083930000648821513800

m_2=010111001010100001100101110110000010101110111011100000011111100011010110101011001001101111101001000010100001111 00
01001101100010111000011100111011110001010100100100111110101101001010010100110111011110000101010111110010001111100110 0
11000010000111111101100001100111001100010101111011101000100011001001110111110110100100101100101000000000001101100111 10
10001111110000001101101001000110000111010110101010101001011010001111111111100110110001000111001010001100100100000000 1
00010101111111000011000101010111001101000010001101001001011111100110000001010000010101010000110001011111110001010010 0
10111010111000100010001101110110111101000000000111001010000011011101101010010101101100000111001010001100101111111101 0
01101010111101000101000001101001000001110010110100110000110100010101000111111100111010001001100100111101001100111000 0
01111001101110100000111011011001011001101000100001000101010000011101110000001011110000000110111001010100111101000010111
00111111000100101100110110100011001100001011000111110110101001111100101101000010110101001000011101010100001011110100 0
00110111111111000111000111100111001000000000101010111011101100101000001000110011100000111001001110111001101011010111 00
00111010011101000101101100110110000111011000000100111110001001110110010011111001111101001100101111100111111001001100 10
10110100111010000100010001001111000011011000100111001001100001001100011011001011110101100110010000001100001100011010 11
10110110100011101110001100011100100010000101101111001001101101100110010100010000100111110000111111011011010101111001 0
11100011011110011010100111111110111011001100011010111110010010001001001010100010111010010011001101101111001110101010 0
01000110111100001000100000000011111000100110100001001101011001000110000001100110110100000110011000010000110111110001 1
01100011011110110010011000011111111101111111001011110001110100110110101100101111111000111001001001000000110010001101 1
11110110001000100110111110001100000010000110110110011000101100000101111000010100111101000110101000110111010110011011 1
11000001010011110011110100111101110100111010100100011111101010000110110110110001111001101111110000100001001000000000 1
10001011011101100111011100111001100111011111001110100100001101011001110101010100010110110100010000011001110101101011 1
10111111110101011011110110011110101100000000000000110101110100011110100011111110100000010110110001101110100000000110010
00001000110011111000011000111000111000110111010011010001001111101100001000000010000000100110000111011100001101010001000
10000111100011000001101110001110111000000101011000110101101100110010101001110001111000111000000100110011010001111001010
00000110101100101110011010101001110110010101101010010101001111011101001011011111111110011111101000000001110111100000 10
01011101011110001011010010101000101010001010101000001110011101100100111000001101010000000010010110000111100000000111100
01001111001101100101000000101111011110101001011101000101001100100011101101111111111011001010011001111100011101111010 11
01110001101010011000110100111100010110111000011101010000100111000000001001010011101100010110011101100011001001011000 10
01001111100011101111010010100001001101101100000011010010100000100111010000110000101001010011000100100001110010000101 0
01000110010101100011000010001110111001001110101110001000101111010111000101101010010000000100111101101000000001010110010

0010010100010001110101010001101001011110001101111100001011101011000010010000010111001101011001110111100110010101111110101111101111010111110001000100001010101111110010110110101000100010100110001011001100010001000110011000111010011001110000000111101010101111010010101010111111101000101001111010111010001100010111100101110110001100100110010000100100001110111001111010101001001110110101111010110110011000110010101101011100001100100111110110101111100111001010000111011011001001110101001101101101000111101010011100011001011111111110111111110100101001111110011010101001111000000110101011011001010000011010000100111100000100111110110100101000111110100110101111111111101001011000111100011001111010111000111101101010011001111001011110100100110101110101011001011010000110101111001010010111111010110101010011011000001101111110001111101000000011

r_2=85161602445832708032841045453165456775816153231248943733634629027 64

s_2=93642832689502321591793798408169459567384257362628912051148325931 41

$nonce_1$=000000000000000000000000000000000000000000000000000000000000000000000000000000000000000010011111101000011010011101100010

$nonce_2$=000000000000000000000000000000000000000000000000000000000000000000000000000000000000000011001110101001101000101101101101

PW_1=0000000085216dc453c650a7a60aed9b6ee5d1ef152a3905b0308a77

PW_2=00000000fba516c7df21a357340ba3107ccc71022479adcaaf21e2aa

pk_2=11696964121675229474652508686465002223910549468927351155818470051892532609444396357094037068654182618944321759466794737798129882568758635294437822529581948884368793908217577164371340761493748973202090386089784117348444896338333045623827263776351530677489423510990401614675498651368806539700892665630794827970074539502519636985635703423904846267873268638027000512869085896386007589288874791941104024208905179471565696860898515742050749627050160402091147050431861907860410853194406578092286681484815311494392050625025947347091600011235379253096668406867851372928876966149816707618821678060737376150795476191572242239444

pk_3=69238631100571491066665417485592019917199099742539341201263231895780674491870379970783473833266001044401042034856804391195001628465483462857299106182212145458379517063944844261333017423470216031155329002706394102255161976236246938997529442889556057234561378726825116615609389426339357733824659845686727507756443831545762362887505137591329563670395004532108723734106376953903841478881633562942554761025134850977453613577924090641747456023771908740547671011951813695009218059625385445239785403796170915414450712050972677065846116720557081721159899303888744110775576320091960341713585593844421344753944737274052328 79866