## 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 *sk1* as a number less than 224 bits. Compute *pki*, *i* = 1, 2, 3.**

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

A screenshot of a social media post

Description automatically generated

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

A screenshot of a cell phone

Description automatically generated

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:

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

Where 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 . Using Fermat’s Little Theorem we get:

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:

A close up of a womans face

Description automatically generated

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:

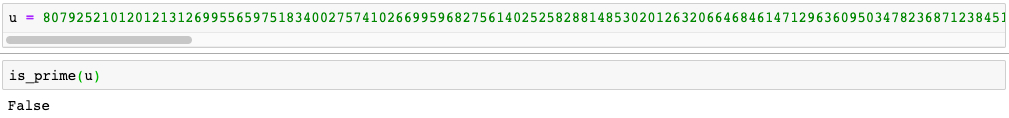


Figure 1.4 – Negative primality test of the factor u returned from the find\_factor method using p - 1

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*:

A screenshot of a social media post

Description automatically generated

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:

A screenshot of a cell phone

Description automatically generated

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

Note that 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 *gk* = 1 (mod *p*). Consequently, Lagrange’s theorem states that q must divide .

A screenshot of a cell phone

Description automatically generated

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 ski as a number less than 224 bits

ski is used in two places – computing the corresponding pki, and in the signing function. In each location, ski 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 ski beyond the requirement does not provide any additional security.

### pki computation

For *i* = 1, 2, 3:

A picture containing clock

Description automatically generated

Figure 1.8 – Function used to compute pki

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 *mi* = *Txi*, *i* = 1, 2, 3.)**

1. **User 1: sign his transaction: i.e, compute the signature over and**

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*, *k1*, *sk1*, and *m1*, and output parameters *r1* and *s1* can be found in the Appendix.

1. **A miner: verify , 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*, *m1*, *r*, and *s* can be found in the Appendix. The output produced by the set of input parameters was True. The intermediate variables *w*, *u1*, *u2*, and *v* can be found in the Appendix.

1. **User 2: sign his transaction *m2* = *Tx2* using the key pair (*sk2*, *pk2*), i.e. executing the same steps as user 1.**

Using the function from part 1) The set of input parameters *p*, *q*, *g*, *k2*, *sk2*, and *m2*, and output parameters *r2* and *s2* 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**

**where \* means any value and *noncei*, *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 *PW1* and *PW2* should be represented as hexadecimal numbers.**

Both *nonce1* and *nonce2* (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 *PW1* and *PW2* (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 and for *PW1*, and and for *PW2*.

1. **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 2192 possible pre-images with 32 leading zeroes out of a total 2224 combinations. Therefore, the probability of finding the correct pre-image with 32 leading zeroes is:

## 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=9891663101749060596110525648800442312262047621700008710332290803354419734415239400374092972505760368555033978883727090878798786527869106102125568674515087767296064898813563305491697474743999164538645162593480340614583420272697669459439956057957775664653137969485217890077966731174553543597150973233536157598924038645446910353512441488171918287556367865699357854285249284142568915079933750257270947667792192723621634761458070065748588907955333315440434095504696037685941392628366404344728480845324408489345349308782555446303365930909965625721154544418491662738796491732039598162639642305389549083822675597763407558360

k1=7636180595255512892709086526638263042018821609914680860810683215268

sk1=4001206756878977378588887638958103884884701595184090082238631140177

m1=011010000111110001110010100000011100100010101111100100111110001000001010010111001000010110101010110011100101101000111110000111001000111011110010101101000000000001101101101100111001100000101010011111011100000110111100011001000000111010101100011111010111001001111010110011101110010110011000111111101001110001000101010010011110111101001001001110101110000100100100101100111010000010110110011000011001000001011101001110111100000111101000011100010110011000000010100000010011110110101010111100100101000111011111011101111011000111100001000100111010011001000000001001110011111101010000001110011110000110101000110010110111110100010000010001010011010001111000101101010110011110000111100101110100011001110101100110100010101111011010101111011111111111101011010100111101010101010100000111000110100001010010010101001100010110000000111100000011110010011111101100000101001010000110100110011010001000110000111110001001001101011011011000010110101010111001100001010000110100001110110111010010110110001010011111100100011110100101100110110001011010010010101101000011011010010011000101010001100001000110000010101110101010110001001000101111000110000111010001001111010101111000000010111010011001100100100110101001110110011011100111110100111111011111001111011110111000111101111001110101110100011000100100110010111111001001101000110111111010100011101110111101010000011110000001111010010001011101001110100000101100000011101010100111000100011010110111101100111011000000011110111000101011011101010001111100100100001010001000001110010100101000011110001000010101110011011110110010011010001001001111110000010011111011111100001000110001011101111101000101001100111110100010000111000000110101111000111110100011110101001011000110011111001011001101101111000110011000100000111001000011001111111111111101110101110110101010011001110110010111001001111000111001010010101000000000000010101000101100001101111101110111010011011001001110011111010010111011000111101110101101110100110001110000111101101010101110011111101011100100111100101110101101111011001111111100110101010001111011101110011000110101110010101000011001011101100000101011101110111000000111111000110101101010110010011011111010010000101000011110001001101100010111000011100111011110001010100100100111110101101001010010100110111011110000101010111110010001111100110011000010000111111101100001100111001100010101111011101000100011001001110111110110100100101100101000000000011011001111010001111110000001101101001000110000111010110101010101001011010001111111111100110110001000111001010001100100100000000100010101111111000011000101010111001101000010001101001001011111100110000001010000010101010000110001011111110001010010010111010111000100010001101110110111101000000000111001010000011011101101010010101101100000111001010001100101111111101001101010111101000101000001101001000001110010110100110000110100010101000111111100111010001001100100111101001100111000001111001101110100000111011011001011001101000100001000101010000011101110000010111100000011011100101010011110100001011100111111000100101100110110100011001100001011000111110110101001111100101101000010110101001000011101010100001011110100000110111111110001110001111001110010000000001010101110111011001010000010001100111000001110010011101110011010110101110000111010011101000101101100110110000111011000001001111100010011101100100111110011111010011001011111001111110010011001010110100111010000100010001001111000011011000100111001001100001001100011011001011110101100110010000011000011000110101110110110100011101110001100011100100010000101101111001001101101100110010100010000100111110000111111011011010101111001011100011011110011010100111111110111011001100011010111110010010001001001010100010111010010011001101101111001110101010001000110111100001000100000000011111000100110100001001101011001000110000001100110110100000110011000010000110111110001101100011011110110010011000011111111101111111001011110001110100110110101100101111111000111001001001000000110010001101111110110001000100110111110001100000010000110110110011000101100000101111000010100111101000110101000110111010110011011111000001010011110011110100111101110100111010100100011111101010000000100

pk1=13190150963760889647094634684573736595322369628268521716277176722504680205306666957427063669382047355005579753986408363017207529087738312031354846179411020392929821489131982024455275601635702544446679012547008732424008115598107948911123181902989407565493889955188285972381676176352955758942149787306114718970006328435881616077569969216032235994360740773429218909181563508361363587117820083370967410980903079772729089192202299692711592106070919451037142106787287918678319935616746224007115660659895996235128698926340666694953849011519321142057025235977157794326767978009471826246937284652336421864460645512691392114275

r1=5102491627416874343359069298372509963208458310973379563071871951255

s1=2231571007439116961204453230930081120395191936038941107639012237189

w=7919988037649650841374863349148472839229621060837749601811945763681

u1=1932139136555447985697411803330180555739694494026775736635882058118

u2=12119820097568046334323666133635880680752059477059327337090107985363

v=5102491627416874343359069298372509963208458310973379563071871951255

k2=1126289620759427337161193756796339715982474853764963010968386403961

sk2=5779569239846031148528787672249249942175134704083930000648821513800

m2=010111001010100001100101110110000010101110111011100000011111100011010110101011001001101111101001000010100001111000100110110001011100001110011101111000101010010010011111010110100101001010011011101111000010101011111001000111110011001100001000011111110110000110011100110001010111101110100010001100100111011111011010010010110010100000000001101100111101000111111000000110110100100011000011101011010101010100101101000111111111110011011000100011100101000110010010000000010001010111111100001100010101011100110100001000110100100101111110011000000101000001010101000011000101111111000101001001011101011100010001000110111011011110100000000011100101000001101110110101001010110110000011100101000110010111111110100110101011110100010100000110100100000111001011010011000011010001010100011111110011101000100110010011110100110011100000111100110111010000011101101100101100110100010000100010101000001110111000001011110000001101110010101001111010000101110011111100010010110011011010001100110000101100011111011010100111110010110100001011010100100001110101010000101111010000011011111111000111000111100111001000000000101010111011101100101000001000110011100000111001001110111001101011010111000011101001110100010110110011011000011101100000100111110001001110110010011111001111101001100101111100111111001001100101011010011101000010001000100111100001101100010011100100110000100110001101100101111010110011001000001100001100011010111011011010001110111000110001110010001000010110111100100110110110011001010001000010011111000011111101101101010111100101110001101111001101010011111111011101100110001101011111001001000100100101010001011101001001100110110111100111010101000100011011110000100010000000001111100010011010000100110101100100011000000110011011010000011001100001000011011111000110110001101111011001001100001111111110111111100101111000111010011011010110010111111100011100100100100000011001000110111111011000100010011011111000110000001000011011011001100010110000010111100001010011110100011010100011011101011001101111100000101001111001111010011110111010011101010010001111110101000011011011011000111110011011111100001000010010000000011000101101110110011101110011100110011101111100111010010000110101100111010101010001011011010001000001100111010110101111011111111010101101111011001111010110000000000000110101110100011110100011111110100000010110110001101110100000001100100000100011001111100001100011100011100011011101001101000100111110110000100000010000001001100001110111000011010100010001000011110001100000110111000111011100000101011000110101101100110010101001110001111000111000001001100110100011110010100000011010110010111001101010100111011001010110101001010100111101110100101101111111111001111110100000001110111100000100101110101111000101101001010100010101000101010100000111001110110010011100000110101000000010010110000111100000001111000100111100110110010100000010111101111010100101110100010100110010001110110111111111011001010011001111100011101111010110111000110101001100011010011110001011011100001110101000010011100000001001010011101100010110011101100011001001011000100100111110001110111101001010000100110110110000001101001010000010011101000011000010100101001100010010000111001000010100100011001010110001100001000111011100100111010111000100010111101011100010110101001000000100111101101000000010101100100010010100010001110101010001101001011110001101111100001011101011000010010000010111001101011001110111100110010101111110101111101111010111110001000100001010101111110010110110101000100010100110001011001100010001000110011000111010011001110000000111101010101111010010101010111111101000101001111010111010001100010111100101110110001100100110010000100100001110111001111010101001001110110101111010110110011000110010101101011100001100100111110110101111100111001010000111011011001001110101001101101101000111101010011100011001011111111110111111110100101001111110011010101001111000000110101011011001010000011010000100111100000100111110110100101000111110100110101111111111101001011000111100011001111010111000111101101010011001111001011110100100110101110101011001011010000110101111001010010111111010110101010011011000001101111110001111101000000011

r2=8516160244583270803284104545316545677581615323124894373363462902764

s2=9364283268950232159179379840816945956738425736262891205114832593141

nonce1=00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000010011111101000011010011101100010

nonce2=00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000011001110101001101000101101101101

PW1=0000000085216dc453c650a7a60aed9b6ee5d1ef152a3905b0308a77

PW2=00000000fba516c7df21a357340ba3107ccc71022479adcaaf21e2aa

pk2=11696964121675229474652508686465002223910549468927351155818470051892532609444396357094037068654182618944321759466794737798129882568758635294437822529581948884368793908217577164371340761493748973202090386089784117348444896338333045623827263776351530677489423510990401614675498651368806539700892665630794827970074539502519636985635703423904846267873268638027000512869085896386007589288874791941104024208905179471565696860898515742050749627050160402091147050431861907860410853194406578092286681484815311494392050625025947347091600011235379253096668406867851372928876966149816707618821678060737376150795476191572242239444

pk3=6923863110057149106666541748559201991719909974253934120126323189578067449187037997078347383326600104440104203485680439119500162846548346285729910618221214545837951706394484426133301742347021603115532900270639410225516197623624693899752944288955605723456137872682511661560938942633935773382465984568672750775644383154576236288750513759132956367039500453210872373410637695390384147888163356294255476102513485097745361357792409064174745602377190874054767101195181369500921805962538544523978540379617091541445071205097267706584611672055708172115989930388874411077557632009196034171358559384442134475394473727405232879866