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 = 161585042024024262539911319503668005514820533991936551228050516576297060402526413293692294259272190069564737424769

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,
 80792521012012131269955659751834002757410266995968275614025258288148530201263206646846147129636095034782368712384519
893943641863398312806338747963782392923265936898340350387358201165266339339704498811349277692481146363231336300996659
753772809134790576619820837861560563416171843727915949442496818464040976140278193081677711641206211580015942455384765
144892595321111739393623341618105978256416706894225642006316565354661148064717778465381920470479616049441594917191871
183780972949256698364365971631232769775454026991957750963849972192971215891213833094419016280605610735416499107060455
47583106995719979463007803486771]
```

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,
 59935218845754763969374017152268083586532218407528662025638671643992102933478299856200831399510384081711695321035964
351144737210122146499565317770665391891163401555563300180177359029202308360455261391865519466992936896268865967354406
188599056669477493808909559794050544343071457319421113422378478474493991138731444089963805601647427060985306314263832
950042903990489732893385841289737425396287831310219916340031965539272302770310135492781437785195434533688009758466959
8515815463134218486896858352351187255794957262790967727451704317515673833695348341]
```

```
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.